Contents

1 Finite Difference Method: A Tutorial 1
1.1 Solving a simple problem with mesh and matrix 1
1.2 Sparse Matrix 2
1.3 Finite Difference 3
1.3.1 Taylor Series 4
1.3.2 Finite Difference formulation for Elliptic Equations 4
1.4 Exercises 5
1.4.1 Homework 5
1.4.1.1 Average-of-Neighbor in a Hexagonal Mesh 5
1.4.1.2 One-dimensional Poisson's Equation 6

2 Finite Volume Discretization 7

3 PDEs of Semiconductor Devices 9
3.1 Shockley's Equations 9
3.1.1 Drift-Diffusion Current 9
3.1.2 Recombination and Generation 10
3.2 Non-linear Elliptical Equations 10
3.3 A Naive Discretization of the Shokley's equations 11

4 Solving Non-linear Equations with Newton's Method 14
4.1 Jacobian Matrix 14
4.2 Non-linear Newton Iteration 15
4.2.1 Worked Example with 2 Unknowns 15
4.3 Graphical Interpretation of the 2D example 17
4.4 Summary 18
4.5 Exercises 18
4.5.1 1D MOS Capacitor 18

5 Overall Structure of a Semiconductor Device Simulator 19
5.1 General-Purpose PDE Applications and Frameworks 20
5.2 Reusable Components 20

6 Discretization Error 22
6.1 Finite-Difference Discretization Schemes 22
6.2 Numerical Diffusion 23
6.3 Fourier Error Analysis 24

7 Scharfetter-Gummel Discretization 26
7.1 Derivation of S-G Discretization 26
7.2 S-G Discretization in Diffusion and Drift limits 28
7.3 S-G Discretization and Artificial Diffusion 29

8 Triangular Mesh for General 2D Geometry 31
8.1 Triangular Mesh 31
8.2 Finite-Volume Method and Voronoi Cell 32
8.3 Assembly of Equations 33
8.4 Exercise 33

9 Boundary Conditions 34
9.1 Outer boundary of the device structure 34
9.1.1 Natural boundary 34
9.1.2 Ohmic contacts 35
9.1.3 Gate contacts 37
9.2 Semiconductor-Insulator Interface 37
9.3 Exercise: 1D Shockley equations (PN junction diode) 39

10 Automatic Differentation 40
10.1 Computation Graph 41
10.2 Forward- and Backward-Accumulation 42
10.3 Operator Overloading 43
10.4 Further Readings 44

11 Circuit Simulation 45
11.1 KCL and Nodal Analysis 45
11.1.1 Independent Voltage Source and Auxiliary Variable 46
11.2 Circuit Elements 46
11.2.1 Resistor 46
11.2.2 Capacitor 46
11.2.3 Inductor 47
11.2.4 Diode 47
11.2.5 MOSFET 48
11.2.6 Others 48
11.3 Summary 48
11.4 Exercise 48

12 Floating-Point Rounding Error 49
12.1 IEEE 754 Standard for Floating-Point Arithmetics 49
12.2 Examples of Rounding Error 50
12.3 Function Evaluation to Machine Precision 51
12.4 High Precision and Adaptive Precision Arithmetics 51
12.5 Variable Scaling 52

13 Linear Solvers and Conditioning 54
13.1 Direct Solvers based on LU Decomposition 54
13.2 Iterative Solvers 54
13.2.1 Condition Number 55
13.3 Direct Solvers Based on Multi-Frontal Method 56
13.4 Condition Number of Shockley Equations 56

A Reading Mathematical Expressions 58

B Some Mathematics Recapitulation 60
B.1 Partial differential equations 60
B.1.1 PDEs, Terminologies and Classifications 60
B.1.1.1 Linear vs non-linear 60
B.1.1.2 homogeneous vs inhomogeneous 60
B.1.1.3 Order 60
B.1.1.4 Boundary conditions 61
B.1.2 Common analytic techniques 61
B.1.2.1 A worked example 61

1 Finite Difference Method: A Tutorial

1.1 Solving a simple problem with mesh and matrix

Let us start with a simple problem. Consider a 50-by-50 mesh with 2500 nodes. We assign a real number to each of the nodes using the following rule:

- Numbers at nodes along the first edge are 10.
- Numbers at nodes along other three edges are all zero.
- For any internal node, the number equals to the average of the numbers at its 4 neighbors.

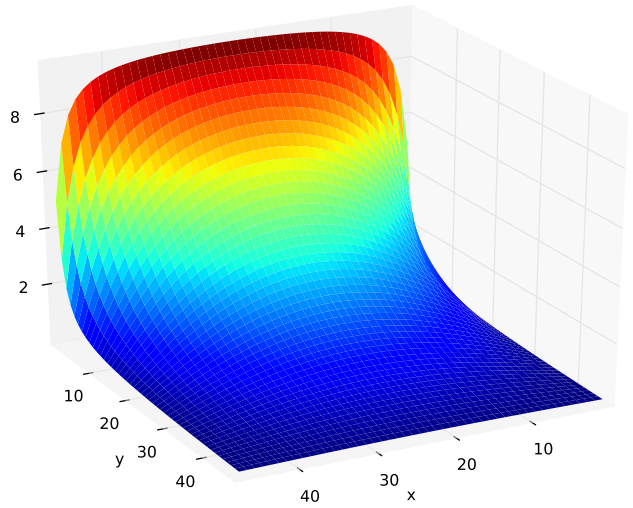


Figure 1.1 Solution to the 2D average-of-neighbor problem, (mesh size 50×50).

Since the values along all edges are known (*boundary conditions*), we only need to determine the values at the 48×48 internal nodes. The solution is plotted in **Figure 1.1**.

To demonstrate the procedure of obtaining this solution, we consider a smaller 6×6 problem, where only $4 \times 4 = 16$ unknowns are to be solved. We first label the unknown nodes as $v_1, v_2, \ldots, v_{16}$,

$$\begin{array}{cccccc} & 10 & 10 & 10 & 10 & \\ 0 & v_1 & v_2 & v_3 & v_4 & 0 \\ 0 & v_5 & v_6 & v_7 & v_8 & 0 \\ 0 & v_9 & v_{10} & v_{11} & v_{12} & 0. \\ 0 & v_{13} & v_{14} & v_{15} & v_{16} & 0 \\ & 0 & 0 & 0 & 0 & \end{array} \tag{1.1}$$

Then we apply the requirement that the number at any node equals to the average of its four neighbors. This requirement can be easily written as an algebraic equation. Taking the node v_6 as an example, we have the equation

$$v_6 = \frac{1}{4}(v_2 + v_5 + v_7 + v_{10}), \tag{1.2}$$

or

$$-v_2 - v_5 + 4v_6 - v_7 - v_{10} = 0. \tag{1.3}$$

Similarly, we write for v_1

$$4v_1 - v_2 - v_5 = 10, \tag{1.4}$$

and for v_2

$$-v_1 + 4v_2 - v_3 - v_6 = 10. \tag{1.5}$$

$$\vdots$$

We repeat the same on all the nodes. ⋆ It is obvious that we have in total 16 equations and 16 unknowns. We write these 16 equations in matrix form

$$\begin{pmatrix}
4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4
\end{pmatrix}
\cdot
\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \\ v_{13} \\ v_{14} \\ v_{15} \\ v_{16} \end{pmatrix}
=
\begin{pmatrix} 10 \\ 10 \\ 10 \\ 10 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{1.6}$$

This set of linear equations can be easily solved. The result is plotted in **Figure 1.2**.

1.2 Sparse Matrix

We plot the *non-zero pattern* of the matrix in **(1.6)** in **Figure 1.3**.

In both cases, we see that most entries in the matrix are zeros, and we call this matrix a *sparse matrix*. For a problem with $N \times N$ mesh, as the problem size N increases, the matrix size grows $\sim N^2 \times N^2$, while the number of non-zero entries grows $\sim N^2$. It is very important to take advantage of this sparse nature of the matrices, otherwise the problem size will soon become unmanageable in both computation time and memory consumption.

We define the *bandwidth* of a row in a sparse matrix as the distance between the first and last (inclusive) non-zero entries of that row. The *bandwidth* of this matrix is the maximum row bandwidth of all its rows.

⋆ you are strongly encouraged to do this once by-hand.

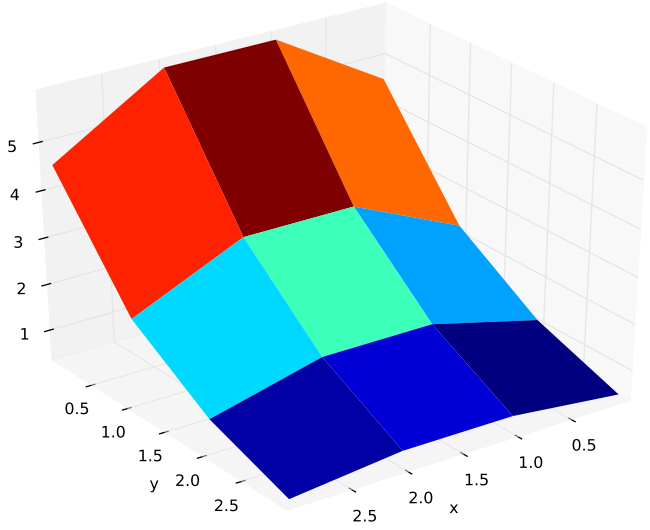
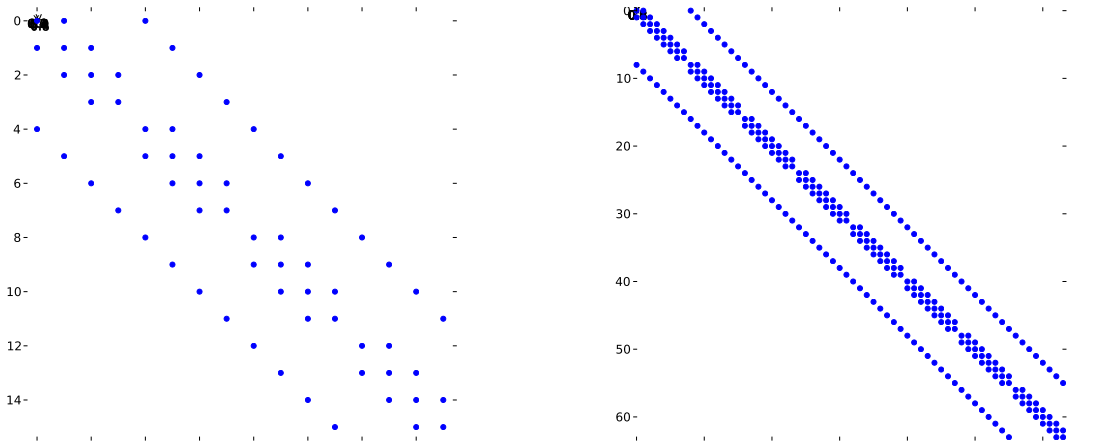


Figure 1.2 Average-of-neighbor problem on a 6×6 mesh.



6×6 average-of-neighbor problem. 10×10 average-of-neighbor problem.

Figure 1.3 Non-zero pattern of the matrix.

1.3 Finite Difference

Recall that the definition of the derivative of function $f(x)$ is,

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}, \tag{1.7}$$

we can obviously approximate this derivative using

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x+h) - f(x)}{h}, \tag{1.8}$$

for a sufficiently small value of h. ★

★ But how good is this approximation? And how do we approximate high-order derivatives?

1 Taylor Series

Expand the function $f(x)$ in the vicinity of $x = a$,

$$\begin{aligned} f(x) &= f(a) + \frac{(x-a)}{1!}f' + \frac{(x-a)^2}{2!}f'' + \dots \\ &= \sum_{n=0}^{\infty} \frac{(x-a)^n}{n!} f^{(n)}(a) \end{aligned}$$

Alternatively, we can write

$$\begin{aligned} f(x+h) &= f(x) + \frac{h}{1!}f'(x) + \frac{h^2}{2!}f''(x) + \dots \\ &= \sum_{n=0}^{\infty} \frac{h^n}{n!} f^{(n)}(x) \end{aligned} \tag{1.9}$$

From (1.9), we have the *forward difference* and *backward difference* approximation to $\frac{\mathrm{d}f}{\mathrm{d}x}$

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f(x+h) - f(x)}{h} + O(h^2), \tag{1.10}$$

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f(x) - f(x-h)}{h} + O(h^2). \tag{1.11}$$

We call $O(h^2)$ ★ the *truncation error* of this finite-difference approximation.

Taking the sum of (1.10) and (1.11), we can get a better approximation (*central difference*),

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f(x+h) - f(x-h)}{2h} + O(h^3). \tag{1.12}$$

Similarly, we can get an approximation to the 2nd order derivative

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2) \tag{1.13}$$

2 Finite Difference formulation for Elliptic Equations

We use the Laplace equation as an example,

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0. \tag{1.14}$$

We shall solve this equation on the mesh grid shown in **Figure 1.4**.

For simplicity, we assume $\Delta x = \Delta y = h$.

Apparently, we need to approximate the Laplacian operator ∇^2 on this discrete mesh grid. We can follow the same procedure in the previous section, and approximate the Laplacian operator as

---

★ read, plus second and higher-order terms.

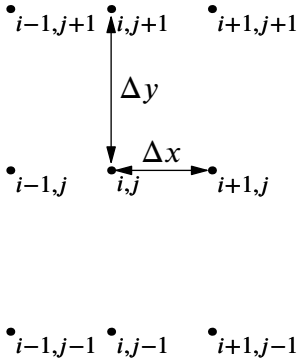


Figure 1.4

$$\nabla^2 u = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{(\Delta x)^2} + \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{(\Delta y)^2}. \tag{1.15}$$

Since $\Delta x = \Delta y = h$, the Laplace equation **(1.14)** can be written as

$$\nabla^2 u_{i,j} = -\frac{1}{h^2}\left[4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}\right] = 0 \tag{1.16}$$

It can be shown that the truncation error of **(1.16)** is $O(h^2)$.

Incidentally, the discrete equation **(1.16)** has the same form as the average-of-neighbor problem described in the previous section.

1.4 Exercises

1.4.1 Homework

1.4.1.1 Average-of-Neighbor in a Hexagonal Mesh ⋆

Consider the hexagonal mesh shown below, where the value at the boundary nodes are given.

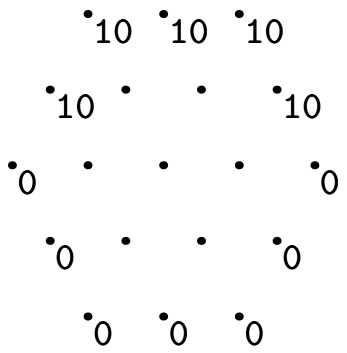


Figure 1.5 Hexagonal Mesh

For each internal node, the value equals to the average of all its neighbors' values.

⋆ 8 Programming Credits

- Write down the matrix equation that solves the problem.
- What is the bandwidth of your matrix.
- If we change the ordering of the nodes in the quadrilateral and the hexagonal mesh, how does the matrix bandwidth change.
- Write a program to solve the hexagonal average-of-neighbor problem of arbitrary mesh size.

1.4.1.2 One-dimensional Poisson's Equation ⋆

Consider the one-dimensional electrostatic problem governed by the Poisson's equation

$$\frac{\mathrm{d}^2 V}{\mathrm{d}x^2} = -\frac{\rho}{\varepsilon}, \tag{1.17}$$

where $0 \leq x \leq 1$. For simplicity, we choose a system of units where $\varepsilon = 1$. The charge density ρ is uniform in this region, and $\rho = 1$. The boundary conditions are $V|_{x=0} = 0$ and $V|_{x=1} = 10$.

- Divide the region into 10 equal segments, and discretize the problem using the 1D finite-difference scheme. Count the number of unknowns and equations.
- Write the system of equations in matrix form. Solve this equation and plot the potential as a function of x.
- Write a program to solve this 1D Poisson's equation with arbitrary number of segments.

⋆ 5 Programming Credits

2 Finite Volume Discretization

An alternative to the finite-difference discretization is the *finite-volume discretization*. As we shall see later, the finite-volume method has several advantages over the finite-difference method in dealing with the semiconductor equations. For instance, the finite-volume method closely relates to conservation laws, and the treatment of boundaries and interfaces is easier as well.

Again we consider the Poisson's equation

$$\nabla^2 \phi = -\frac{\rho}{\varepsilon}, \tag{2.1}$$

which is one of the Maxwell's equations ★

$$\nabla \cdot \vec{D} = \rho. \tag{2.2}$$

Recalling the Gauss Theorem,

$$\int_V \nabla \cdot \vec{A}\, \mathrm{d}V = \oint_S \vec{A} \cdot \overrightarrow{\mathrm{d}s}, \tag{2.3}$$

where S is the closed surface that surrounds the volume V. We prefer to write the Poisson's equation in the integral form.

$$\oint_S \vec{D} \cdot \overrightarrow{\mathrm{d}s} = \int_V \rho \mathrm{d}V \tag{2.4}$$

Our task is to express this integral equation at each of the discrete mesh nodes, so that we can solve it as we did in the previous chapter.

We consider a cell in the uniform rectangular 2D mesh shown in **Figure 2.1**.

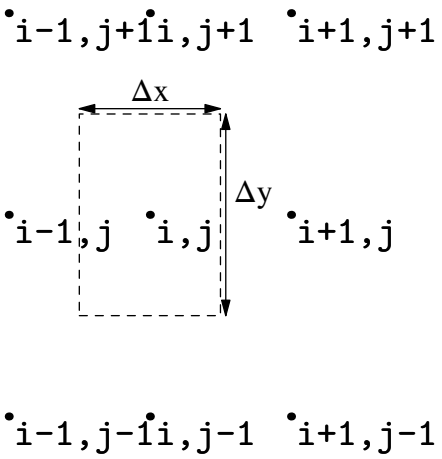


Figure 2.1

Each cell has width Δx and height Δy, and an imaginary depth Δz. The integration runs over the dotted surface S. We have

$$\rho_{i,j} \cdot \Delta x \cdot \Delta y \cdot \Delta z = \varepsilon \frac{\phi_{i,j} - \phi_{i-1,j}}{\Delta x} \cdot \Delta y \cdot \Delta z + \varepsilon \frac{\phi_{i,j} - \phi_{i+1,j}}{\Delta x} \cdot \Delta y \cdot \Delta z + \varepsilon \frac{\phi_{i,j} - \phi_{i,j-1}}{\Delta y} \cdot \Delta x \cdot \Delta z + \varepsilon \frac{\phi_{i,j} - \phi_{i,j+1}}{\Delta y} \cdot \Delta x \cdot \Delta z \tag{2.5}$$

★ $\vec{D} = \varepsilon \vec{E}$

where $\phi_{i,j}$ is the unknown potential and ρ the known charge density. We therefore have one unknown and one equation at each node. Assuming $\Delta x = \Delta y = h$,

$$\varepsilon \left(4\phi_{i,j} - \phi_{i+1,j} - \phi_{i-1,j} - \phi_{i,j+1} - \phi_{i,j-1}\right) = h^2 \cdot \rho_{i,j}. \tag{2.6}$$

Therefore, for the Poisson's equation, the finite-difference and finite-volume methods are equivalent.

It can be shown that the system of equations can be written as

$$\boldsymbol{A} \cdot \begin{pmatrix} \phi_{1,1} \\ \phi_{1,2} \\ \vdots \\ \phi_{N,N} \end{pmatrix} = \begin{pmatrix} b_{1,1} \\ b_{1,2} \\ \vdots \\ b_{N,N} \end{pmatrix}$$

where $\boldsymbol{A}$ is an $N^2 \times N^2$ sparse matrix. The vector $\boldsymbol{b}$ contain the total electrical charge in each cell. For boundary nodes, the corresponding element in $\boldsymbol{b}$ contains the boundary condition as well. We shall discuss the various boundary conditions in a later chapter. For the moment, we say that the potential vector $\boldsymbol{\phi}$ can be easily solved by inverting $\boldsymbol{A}$

$$\boldsymbol{\phi} = \boldsymbol{A}^{-1}\boldsymbol{b}.$$

3 PDEs of Semiconductor Devices

3.1 Shockley's Equations

The following set of equations are commonly referred to as the Shockley Equations for semiconductor devices,

$$\nabla^2\phi = -\frac{q}{\varepsilon}(N_{\mathrm{D}} - N_{\mathrm{A}} + p - n) \tag{3.1}$$

$$\frac{\partial n}{\partial t} = \frac{1}{q}\nabla \cdot \vec{J}_n + G \tag{3.2}$$

$$\frac{\partial p}{\partial t} = -\frac{1}{q}\nabla \cdot \vec{J}_p + G, \tag{3.3}$$

ϕ	=	electrostatic potential	V
n	=	electron concentration	cm^{-3}
p	=	hole concentration	cm^{-3}
$\vec{J}_n$	=	electron particle current	Acm^{-2}
$\vec{J}_p$	=	hole particle current	Acm^{-2}
G	=	net carrier generation	$\mathrm{cm}^{-3}\mathrm{s}^{-1}$
q	=	electronic charge	C
ε	=	permittivity	Fcm^{-1}
N_{D}	=	ionized donor concentration	cm^{-3}
N_{A}	=	ionized acceptor concentration	cm^{-3}

The first equation **(3.1)** is the Poisson's equation ★ for electrostatics, which is part of the Maxwell's equations. The continuity equations **(3.2)** and **(3.3)** actually says the conservation of particles. ★★

One notes that there are more unknowns than there are equations. In order to get a solution, we need three more equations that relates $\vec{J}_n$, $\vec{J}_p$ and G to ϕ, n and p. We will describe the additional equations in the following two sections.

3.1.1 Drift-Diffusion Current

The movement of carriers produces the particle currents in semiconductor. There are two mechanisms by which carriers move in the crystal, namely *drift* and *diffusion*. ★★★ Under certain simplifying conditions, the drift and diffusion components in electron and hole currents can be expressed as,

$$\vec{J}_n = q\mu_n n\vec{E}_n + qD_n\nabla n, \tag{3.4}$$

$$\vec{J}_p = q\mu_p p\vec{E}_p - qD_p\nabla p. \tag{3.5}$$

μ_n	=	electron mobility	$\mathrm{cm}^2\mathrm{V}^{-1}\mathrm{s}^{-1}$
μ_p	=	hole mobility	$\mathrm{cm}^2\mathrm{V}^{-1}\mathrm{s}^{-1}$
D_n	=	electron diffusivity	$\mathrm{cm}^2\mathrm{s}^{-1}$
D_p	=	hole diffusivity	$\mathrm{cm}^2\mathrm{s}^{-1}$

★ Apparently, we do not consider the effect of magnetic fields on the device.

★★ However, conservation of energy or momentum is not guaranteed.

★★★ We shall discuss these assumptions in details in a later lecture.

$\vec{E}_n$ = electron driving force Vcm^{-1}
$\vec{E}_p$ = hole driving force Vcm^{-1}

For simplicity, we assume $\vec{E}_n = \vec{E}_p = \vec{E} = \nabla\phi$.

The carrier mobility and diffusivity are related by the Einstein's relation,

$$D_n = \frac{kT}{q}\mu_n \tag{3.6}$$

$$D_p = \frac{kT}{q}\mu_p. \tag{3.7}$$

3.1.2 Recombination and Generation

Carriers can be added or removed to the conduction or valence band in various processes. In generation and recombination processes, a pair of electron and hole is generated or removed at once. Important R-G processes are

- Shockley-Read-Hall recombination
- Auger recombination
- Direct recombination
- Avalanche generation
- Band-to-band tunneling

The physics of many generation processes are rather complex. For simplicity, we first consider the most common SRH recombination. Assuming that the traps are located at the center of the band-gap, the SRH generation rate is

$$G = \frac{{n_i}^2 - np}{\tau_n(n + n_i) + \tau_p(p + n_i)}. \tag{3.8}$$

n_i = intrinsic carrier concentration cm^{-3}
τ_n = electron minority carrier life-time s
τ_p = hole minority carrier life-time s

3.2 Non-linear Elliptical Equations

By expanding the Nabla operator in Cartesian coordinates, we already know that the Poisson's equation is an elliptical equation.

Substituting (3.4) into (3.2), we obtain

$$\begin{aligned}\frac{\partial n}{\partial t} &= \left(\mu_n n\nabla\cdot\nabla\phi + D_n\nabla\cdot\nabla n\right)\\ &= \left(\mu_n n\nabla^2\phi + D_n\nabla^2 n\right).\end{aligned}$$

Using the determinant, we see that the continuity equations are also elliptical.

It is apparent that the drift current term as well as the SRH generation rate term are non-linear. We shall learn the techniques of solving the non-linear PDEs in the next chapter, ★ while in the rest of this chapter, we shall follow our existing paradigm for linear equations, i.e., discretizing the problem on a mesh, and write down the equation at each node.

3.3 A Naive Discretization of the Shokley's equations

We prefer the finite-volume discretization, and thus use the integration form of the Shockley's equations, ★★

$$\oint_S \vec{D} \cdot \overrightarrow{\mathrm{d}s} = \int_V (N_\mathrm{D} - N_\mathrm{A} + p - n)V \tag{3.9}$$

$$\frac{1}{q} \oint_S \vec{J}_n \cdot \overrightarrow{\mathrm{d}s} = \int_V (\frac{\partial n}{\partial t} - G)\,\mathrm{d}V \tag{3.10}$$

$$\frac{1}{q} \oint_S \vec{J}_p \cdot \overrightarrow{\mathrm{d}s} = \int_V (-\frac{\partial p}{\partial t} + G)\,\mathrm{d}V \tag{3.11}$$

One can discretize these equations using finite-volume method, as in the previous chapter, but we shall limit ourselves to the steady-state case where $\frac{\partial n}{\partial t} = \frac{\partial p}{\partial t} = 0$. For each node (i,j), we have three variables, potential $\phi_{i,j}$, electron density $n_{i,j}$ and hole density $p_{i,j}$.

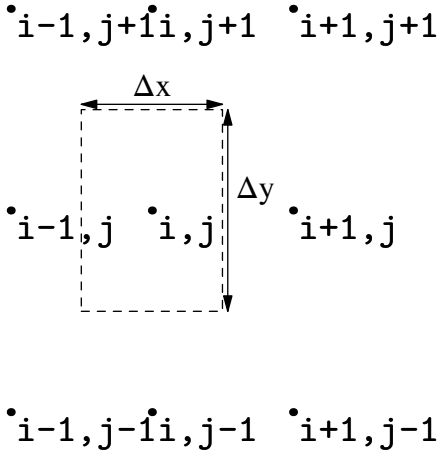


Figure 3.1

Poisson's equation:

$$\begin{aligned} q(p_{i,j} - n_{i,j} + N_\mathrm{D} - N_\mathrm{A}) \cdot \Delta x \cdot \Delta y = \varepsilon \frac{\phi_{i,j} - \phi_{i-1,j}}{\Delta x} \cdot \Delta y + \varepsilon \frac{\phi_{i,j} - \phi_{i+1,j}}{\Delta x} \cdot \Delta y + \\ \varepsilon \frac{\phi_{i,j} - \phi_{i,j-1}}{\Delta y} \cdot \Delta x + \varepsilon \frac{\phi_{i,j} - \phi_{i,j+1}}{\Delta y} \cdot \Delta x. \end{aligned} \tag{3.12}$$

★ We shall see in the next chapter that we just need to add one technique to our toolbox to solve the non-linear equations.

★★ Note how **(3.10)** spells out the conservation law very explicitly.

Electron continuity equation:

$$
\begin{aligned}
-\frac{{n_i}^2 - n_{i,j}p_{i,j}}{\tau_n(n_{i,j}+n_i)+\tau_p(p_{i,j}+n_i)}\cdot\Delta x\cdot\Delta y = {} & \mu_n\frac{n_{i,j}+n_{i-1,j}}{2}\frac{\phi_{i,j}-\phi_{i-1,j}}{\Delta x}\cdot\Delta y + \\
& \mu_n\frac{n_{i,j}+n_{i+1,j}}{2}\frac{\phi_{i,j}-\phi_{i+1,j}}{\Delta x}\cdot\Delta y + \\
& \mu_n\frac{n_{i,j}+n_{i,j-1}}{2}\frac{\phi_{i,j}-\phi_{i,j-1}}{\Delta x}\cdot\Delta x + \\
& \mu_n\frac{n_{i,j}+n_{i,j+1}}{2}\frac{\phi_{i,j}-\phi_{i,j+1}}{\Delta x}\cdot\Delta x + \\
& D_n\frac{n_{i-1,j}-n_{i,j}}{\Delta x}\cdot\Delta y + \\
& D_n\frac{n_{i+1,j}-n_{i,j}}{\Delta x}\cdot\Delta y + \\
& D_n\frac{n_{i,j-1}-n_{i,j}}{\Delta x}\cdot\Delta x + \\
& D_n\frac{n_{i,j+1}-n_{i,j}}{\Delta x}\cdot\Delta x.
\end{aligned}
\tag{3.13}
$$

Hole continuity equation:

$$
\begin{aligned}
\frac{{n_i}^2 - n_{i,j}p_{i,j}}{\tau_n(p_{i,j}+p_i)+\tau_p(p_{i,j}+p_i)}\cdot\Delta x\cdot\Delta y = {} & \mu_p\frac{p_{i,j}+p_{i-1,j}}{2}\frac{\phi_{i,j}-\phi_{i-1,j}}{\Delta x}\cdot\Delta y + \\
& \mu_p\frac{p_{i,j}+p_{i+1,j}}{2}\frac{\phi_{i,j}-\phi_{i+1,j}}{\Delta x}\cdot\Delta y + \\
& \mu_p\frac{p_{i,j}+p_{i,j-1}}{2}\frac{\phi_{i,j}-\phi_{i,j-1}}{\Delta x}\cdot\Delta x + \\
& \mu_p\frac{p_{i,j}+p_{i,j+1}}{2}\frac{\phi_{i,j}-\phi_{i,j+1}}{\Delta x}\cdot\Delta x + \\
& D_p\frac{p_{i,j}-p_{i-1,j}}{\Delta x}\cdot\Delta y + \\
& D_p\frac{p_{i,j}-p_{i+1,j}}{\Delta x}\cdot\Delta y + \\
& D_p\frac{p_{i,j}-p_{i,j-1}}{\Delta x}\cdot\Delta x + \\
& D_p\frac{p_{i,j}-p_{i,j+1}}{\Delta x}\cdot\Delta x.
\end{aligned}
\tag{3.14}
$$

Each node is associated with three variable and three equations, we thus write the solution vector on a $N\times N$ mesh as $v = (\phi_{1,1},n_{1,1},p_{1,1},\phi_{1,2},n_{1,2},p_{1,2},\ldots,\phi_{N,N},n_{N,N},p_{N,N})^T$, which has $3\times N\times N$ elements.

This simple discretization is correct, and works in certain cases. However, it requires very fine mesh grids, or it will become numerically unstable. ★ We shall introduce a more sophisticated and robust discretization scheme in a later chapter.

★ Examine **(3.13)** closely, see what are the assumptions that are not accurate.

We wish to express the set of $3 \times N \times N$ equations in matrix form, as was done previously. However, we now face a serious problem: these equations are non-linear.

4 Solving Non-linear Equations with Newton's Method

While a linear system of equations can be generally expressed as

$$\boldsymbol{A}\boldsymbol{v} = \boldsymbol{b}, \tag{4.1}$$

we can use a more general expression

$$\boldsymbol{F}(\boldsymbol{v}) = \boldsymbol{0}. \tag{4.2}$$

We shall discuss the techniques to solve **(4.2)** when the function $\boldsymbol{F}$ is non-linear.

For a non-linear scalar function $f(x)$, we recall that the Newton's iterative method can be used to solve the equation $f(x) = 0$.

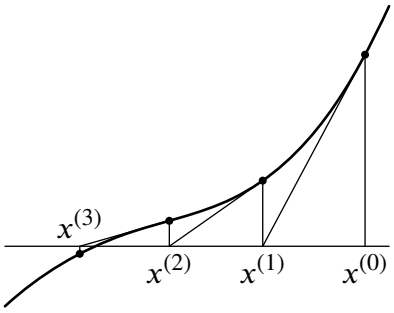


Figure 4.1

As graphically illustrated in **Figure 4.1**, we start with an initial guess $x^{(0)}$, and iterate using

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)} \tag{4.3}$$

to obtain increasingly more accurate solution. A similar technique can be used here for the vector equation $\boldsymbol{F}(\boldsymbol{v}) = 0$, where we generalize the method from one-variable to multi-variable cases.

4.1 Jacobian Matrix

First we need to introduce a new concept for vector function $\boldsymbol{F}$ that corresponds to the derivatives of a scalar function. This new notion is called the Jacobian matrix.

We first write the function $\boldsymbol{F}(\boldsymbol{x})$ in the form

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} = \boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} F_1(x1,x2,x3,\ldots,x_n) \\ F_2(x1,x2,x3,\ldots,x_n) \\ F_3(x1,x2,x3,\ldots,x_n) \\ \cdots \\ F_n(x1,x2,x3,\ldots,x_n) \end{pmatrix}. \tag{4.4}$$

The Jacobian of the function $\boldsymbol{F}(\boldsymbol{x})$ is defined as

$$\boldsymbol{J} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{pmatrix}. \tag{4.5}$$

Note that $\boldsymbol{J}$ is a function of the $\boldsymbol{x}$ vector. The input to the function is a vector, while the output is a matrix. As a matrix, the Jacobian is a linear operator.

It can be shown that for a ``small vector'' $\boldsymbol{p}$, ★

$$\boldsymbol{F}(\boldsymbol{x}+\boldsymbol{p}) = \boldsymbol{F}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x}) \cdot \boldsymbol{p} + o(\|\boldsymbol{p}\|). \tag{4.6}$$

One can easily see the resemblance between Jacobian matrices and function derivatives.

4.2 Non-linear Newton Iteration

In order to solve $\boldsymbol{F}(\boldsymbol{v}) = \boldsymbol{0}$ starting from an initial guess $\boldsymbol{v}^{(0)}$, we use the iteration relation

$$\boldsymbol{v}^{(k+1)} = \boldsymbol{v}^{(k)} - \left\{\boldsymbol{J}[\boldsymbol{v}^{(k)}]\right\}^{-1} \boldsymbol{F}[\boldsymbol{v}^{(k)}]. \tag{4.7}$$

where $\boldsymbol{J}$ is the Jacobian matrix of $\boldsymbol{F}$. ★★

We call the second term in (4.7)

$$\boldsymbol{p}^{(k)} = \left\{\boldsymbol{J}[\boldsymbol{v}^{(k)}]\right\}^{-1} \boldsymbol{F}[\boldsymbol{v}^{(k)}] \tag{4.8}$$

the search direction vector at step k.

4.2.1 Worked Example with 2 Unknowns

Let's look at one example with 2 unknowns. Let

$$\boldsymbol{F}\left[\begin{pmatrix} x \\ y \end{pmatrix}\right] = \begin{pmatrix} x^2+y^2 \\ x^2-y^2 \end{pmatrix},$$

and we shall solve for $\boldsymbol{v}$ such that $\boldsymbol{F}(\boldsymbol{v}) = \boldsymbol{0}$.

Obviously the correct solution is $x = y = 0$. But for illustration purpose let us start from the initial guess $x = 1$, $y = -2$. We first calculate $\boldsymbol{J}$,

$$\boldsymbol{J} = \begin{pmatrix} \frac{\partial(x^2+y^2)}{\partial x} & \frac{\partial(x^2+y^2)}{\partial y} \\ \frac{\partial(x^2-y^2)}{\partial x} & \frac{\partial(x^2-y^2)}{\partial y} \end{pmatrix} = \begin{pmatrix} 2x & 2y \\ 2x & -2y \end{pmatrix}.$$

★ the *norm* $\|\boldsymbol{p}\| = \sqrt{p_1^2 + p_2^2 + \ldots + p_n^2}$ is small.

★★ Compare (4.7) with (4.3).

Then, starting with $\boldsymbol{v}^{(0)} = (1, -2)^T$, we repeatedly apply the iteration **(4.7)**, ★

$$
\begin{aligned}
\boldsymbol{v}^{(1)} &= \begin{pmatrix} 1 \\ -2 \end{pmatrix} - \begin{pmatrix} 2 & -4 \\ 2 & 4 \end{pmatrix}^{-1} \begin{pmatrix} 5 \\ -3 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ -2 \end{pmatrix} - \begin{pmatrix} 0.25 & 0.25 \\ -0.125 & -0.125 \end{pmatrix} \begin{pmatrix} 5 \\ -3 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ -2 \end{pmatrix} - \begin{pmatrix} 0.5 \\ -1 \end{pmatrix} \\
&= \begin{pmatrix} 0.5 \\ -1 \end{pmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{v}^{(2)} &= \begin{pmatrix} 0.5 \\ -1 \end{pmatrix} - \begin{pmatrix} 1 & -2 \\ 1 & 2 \end{pmatrix}^{-1} \begin{pmatrix} 1.25 \\ -0.75 \end{pmatrix} \\
&= \begin{pmatrix} 0.5 \\ -1 \end{pmatrix} - \begin{pmatrix} 0.5 & 0.5 \\ -0.25 & 0.25 \end{pmatrix} \begin{pmatrix} 1.25 \\ -0.75 \end{pmatrix} \\
&= \begin{pmatrix} 0.5 \\ -1 \end{pmatrix} - \begin{pmatrix} 0.25 \\ -0.5 \end{pmatrix} \\
&= \begin{pmatrix} 0.25 \\ -0.5 \end{pmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{v}^{(3)} &= \begin{pmatrix} 0.25 \\ -0.5 \end{pmatrix} - \begin{pmatrix} 0.5 & -1 \\ 0.5 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0.3125 \\ -0.1875 \end{pmatrix} \\
&= \begin{pmatrix} 0.25 \\ -0.5 \end{pmatrix} - \begin{pmatrix} 1 & 1 \\ -0.5 & 0.5 \end{pmatrix} \begin{pmatrix} 0.3125 \\ -0.1875 \end{pmatrix} \\
&= \begin{pmatrix} 0.25 \\ -0.5 \end{pmatrix} - \begin{pmatrix} 0.125 \\ -0.25 \end{pmatrix} \\
&= \begin{pmatrix} 0.125 \\ -0.25 \end{pmatrix}
\end{aligned}
$$

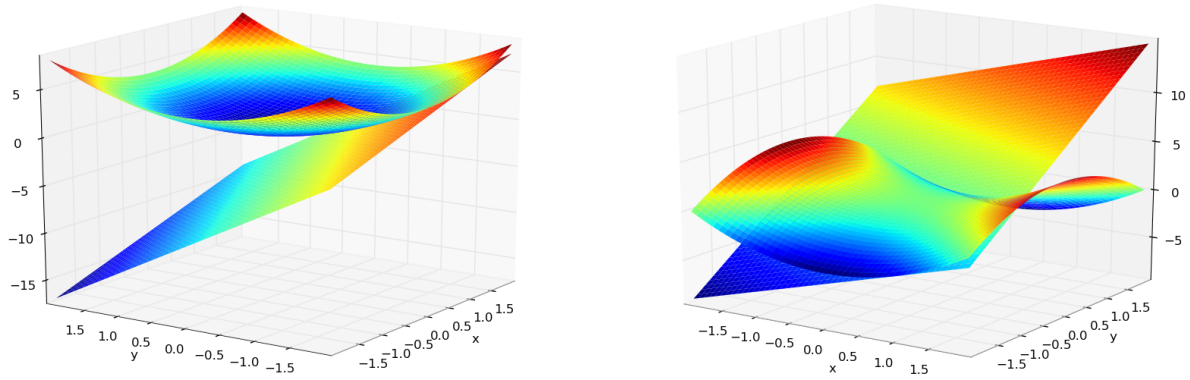
Repeating the above procedure obviously leads us towards the desired solution.

★

$$
\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}
$$

4.3 Graphical Interpretation of the 2D example

Now let us try to visualize what we have been doing. It is obvious that $z_1 = F_1(x,y)$ $z_2 = F_2(x,y)$ defines an elliptic paraboloid and a hyperbolic paraboloid surface, respectively, which are plotted in **Figure 4.2**.



a) Elliptic paraboloid defined by $z_1 = F_1(x,y) = x^2 + y^2$ and the tangential plane corresponding to the initial guess $\boldsymbol{v}^{(o)}$.

b) Hyperbolic paraboloid defined by $z_2 = F_2(x,y) = x^2 - y^2$ and the tangential plane corresponding to the initial guess $\boldsymbol{v}^{(o)}$.

Figure 4.2

The initial guess $\boldsymbol{v}^{(0)}$ defines a point $\vec{p} = (1,-2,5)^T$ on the z_1 surface. The partial derivatives in the Jacobian matrix $\frac{\partial F_1}{\partial x} = 2$ and $\frac{\partial F_1}{\partial y} = -4$ defines two vectors $\vec{u} = (1,0,2)^T$ and $\vec{v} = (0,1,-4)^T$ that are tangential to z_1 surface at point $\vec{p}$. From the one point and two vectors, we obtain the equation for the tangential plane P_1

$$-2(x-1) + 4(y+2) + (z-5) = 0. \tag{4.9}$$

Similarly, we can find the plane P_2, which is tangential to the surface z_2 at the point corresponding to the initial guess. The equation for P_2 is

$$-2(x-1) - 4(y+2) + (z+3) = 0. \tag{4.10}$$

Our trial solution of the next iteration $\boldsymbol{v}^{(1)}$ lies on the intersection of the tangent plane P_1 with the plane $z = 0$. This intersection line is easily found to be

$$2x - 4y - 5 = 0. \tag{4.11}$$

Similarly, $\boldsymbol{v}^{(1)}$ also lies on the intersection of P_2 with the X-Y plane, and the intersection line is

$$2x + 4y + 3 = 0. \tag{4.12}$$

To satisfy both **(4.11)** and **(4.12)**, we must have $\boldsymbol{v}^{(1)} = (0.5,-1)$.

Obviously, the above interpretation is an extension of the 1D example in **Figure 4.1**, with the tangent line replaced by tangent planes here. Equations with more unknowns can be interpreted with planes in higher dimension space, but is difficult to visualize.

4.4 Summary

We have derived the system of discretized Shockley's equations **(3.9)** - **(3.11)**, we have also developed the iterative technique **(4.7)**. In principle, we are now able to solve this set of equations.

However, there remain a few practical problems before we can make the program work correctly and robustly:

- An overall design of the program that links all the components together.
- The proper boundary conditions at ohmic contacts, material interfaces, and the outer boundary of the device structure.
- A better discretization scheme that is more robust than our naive attempt.
- An efficient way to compute the function $\boldsymbol{F}$ and more importantly the Jacobian $\boldsymbol{J}$.

We shall address all the above problems in the following chapters.

4.5 Exercises

4.5.1 1D MOS Capacitor ⋆

For a MOS capacitor in thermal equilibrium, the carrier concentration can be written as

$$n = n_0 \exp\left(\frac{q\psi}{kT}\right)$$
$$p = p_0 \exp\left(-\frac{q\psi}{kT}\right),$$

where n_0 and p_0 are the equilibrium carrier concentrations, and ψ is the band-bending. With this, **(2.1)** can be solved.

For a MOS capacitor with uniform p-type substrate doping of $N_a = 1 \times 10^{17}$ cm^{-3}, and surface potential of $\psi_s = 0.6$ V,

- Discretize this 1D Poisson equation using finite-volume method. Note that the volumes of the first and the last cell are different from the rest.
- Write down the system of non-linear equations; take care of the boundary conditions.
- Write down the Jacobian matrix of this non-linear equation. What is the bandwidth of this matrix.
- Write a program to solve this 1D MOS capacitor problem.

⋆ 10 programming credits

5 Overall Structure of a Semiconductor Device Simulator

We start by describing the general structure of semiconductor device simulators.

At the top level, the components and control flow of a typical simulator is shown in **Figure 5.1**.

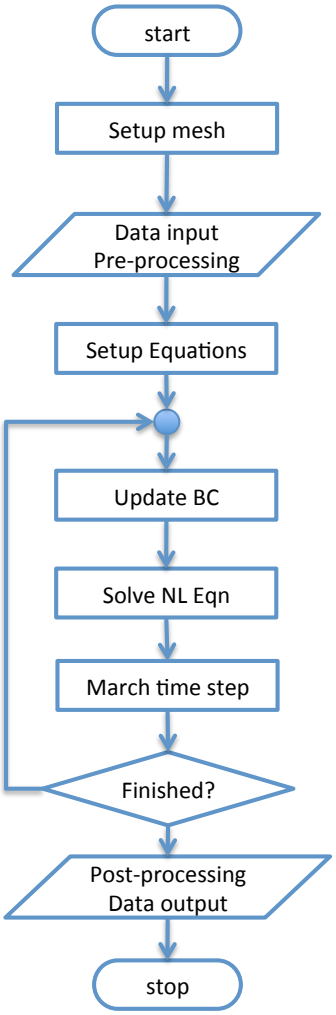


Figure 5.1 Flowchart of a general PDE based application.

At the core of the simulator, we have the non-linear PDE solver, which solves the system of nonlinear equations

$$\boldsymbol{F}(\boldsymbol{v}) = \boldsymbol{0}. \tag{5.1}$$

Newton's method, as described in the last chapter, is most often used as the non-linear solver. The flow-chart is shown in **Figure 5.2**.

We have learned the essential methods of solving the PDEs governing the semiconductor devices. We are ready to write our own simulator.

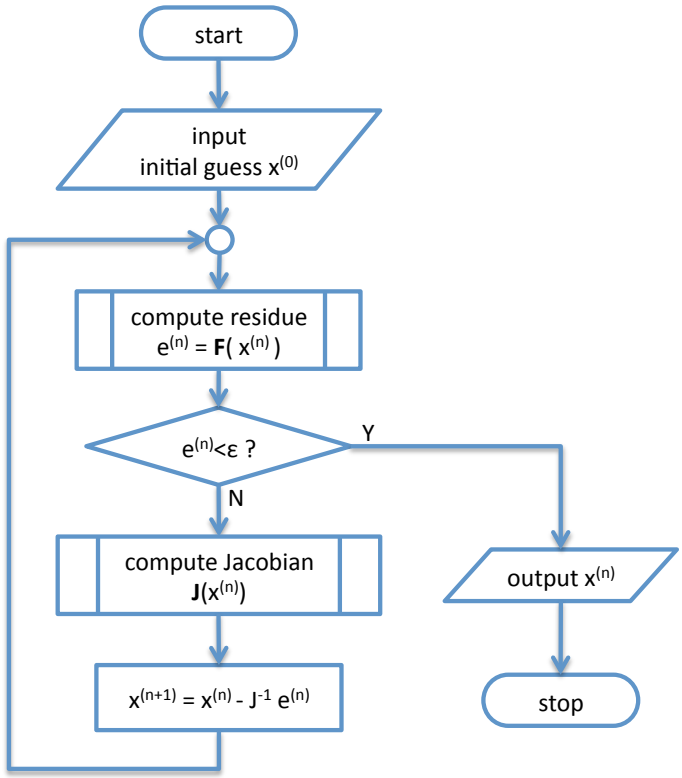


Figure 5.2 Flowchart of a general non-linear PDE solver.

5.1 General-Purpose PDE Applications and Frameworks

COMSOL multi-physics is a commercial application that allows a user to specify custom differential equations and boundary conditions through a GUI. It then solves the PDE and also provides data visualization tools.

Libmesh, on the other hand, provides a framework for programmers to develop PDE application in C++. Many common tasks, such as pre-processing/post-processing, setting-up of the mesh data structure and abstraction of equations/boundary conditions, are already implemented in the framework. The application developer can then focus on the physics itself.

5.2 Reusable Components

- Optimized mathematics library
- Optimized linear algebra library (BLAS, LAPACK, ScaLAPACK, etc.)
- Sparse linear solver and pre-conditioners
  - Direct solver: MUMPS, UMFPack, SuperLU, etc
  - Krylov-space iterative solvers (...)

  - Pre-conditioners (...)
- Nonlinear equation framework (PETsc)
- Finite-difference, finite-element PDE frameworks.
- Computational geometry
- Mesh generator (Delaunay, advancing front, etc.)
- Data interpolation
- Visualization
- Data import/export (HDF and others)

6 Discretization Error

If we implement our device simulator using the naive finite-difference/finite-volume discretization described in the previous chapters, the simulator would work in some simple cases. However, it will face stability problem with most useful devices, such as PN junction diodes. That simple discretization scheme introduces too much harmful discretization error, and would require very fine mesh grid. In this chapter, we discuss why this happens.

Consider the linear convection equation equation

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \tag{6.1}$$

on a domain extending from $-\infty$ to ∞.

We can solve this equation by separation of variable. Write one solution as

$$u_\kappa(x,t) = f(t)\mathrm{e}^{\mathrm{j}\kappa x}, \tag{6.2}$$

so $f(t)$ has to satisfy

$$\frac{\mathrm{d}f}{\mathrm{d}t} = -\mathrm{j}a\kappa f. \tag{6.3}$$

Therefore,

$$u_\kappa(x,t) = C_\kappa \mathrm{e}^{\mathrm{j}\kappa(x-at)}, \tag{6.4}$$

where κ is the wavenumber, and C_κ is a constant. When both κ and a are real, this solution represents a propagating sinusoidal wave. The wave speed is $c = a$. In general, we need to linearly combine many **(6.4)** of different κ, to form the particular solution that satisfies the initial condition. ★

We know very well how this exact solution should behave. We shall solve the same equation numerically, and check if the numerical solution matches the exact solution.

6.1 Finite-Difference Discretization Schemes

There are many ways to approximate the partial derivative using finite-differences. Here we look at three simple schemes.

Central difference:

$$\frac{u_i^{(n+1)} - u_i^{(n)}}{\Delta t} = a\frac{u_{i+1}^{(n)} - u_{i-1}^{(n)}}{2\Delta x} \tag{6.5}$$

First-order up-wind:

$$\frac{u_i^{(n+1)} - u_i^{(n)}}{\Delta t} = a\frac{u_i^{(n)} - u_{i-1}^{(n)}}{\Delta x} \tag{6.6}$$

Second-order up-wind:

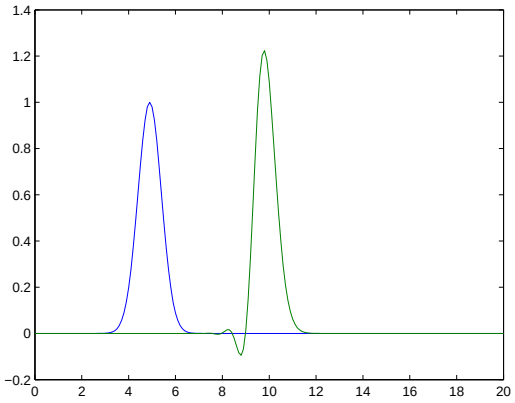
★ Note that wave speed is constant for all wavenumbers.

$$\frac{u_i^{(n+1)} - u_i^{(n)}}{\Delta t} = a\frac{3u_i^{(n)} - 4u_{i-1}^{(n)} + u_{i-2}^{(n)}}{2\Delta x} \tag{6.7}$$

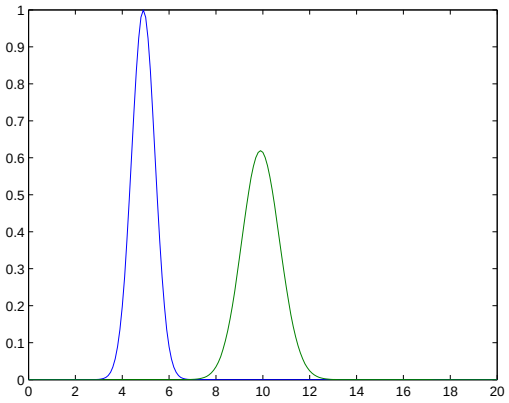
We solve the equation using all the three schemes, and the numerical solutions are plotted in **Figure 6.1**.

We expect the wave packet to shift by a distance of 5, with its shape unchanged. However, in all the three cases, the numerical solution shows severe distortion. In addition to the distortion, the peak position of the shifted waveform is different in the three cases. The obtained wave speed seems to be inconsistent.

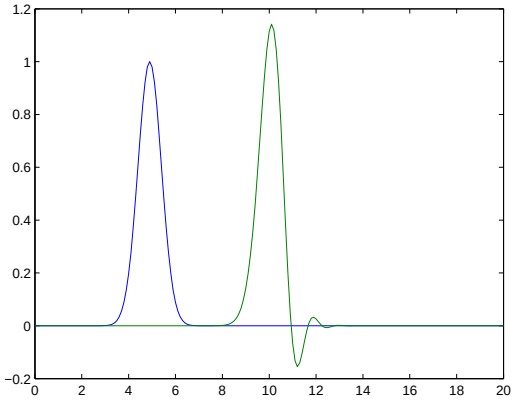
If we double the wave speed, and solve it again in the central difference scheme **Figure 6.2**, the distortion is so severe that the waveform is lost, and oscillation occurs.



Central difference



First-order up-wind



Second-order up-wind

Figure 6.1 Solution of the linear wave equation with $a = 0.2$. The waveform at $t = 0$ is a gaussian wave packet centered at $x = 5$ with $\sigma = 0.5$. Plotted are the initial waveform and the waveform at $t = 25$. Grid size $\Delta x = 0.1$, $\Delta t = 0.1$.

6.2 Numerical Diffusion

We examine the first-order up-wind scheme more closely, recall the Taylor expansion

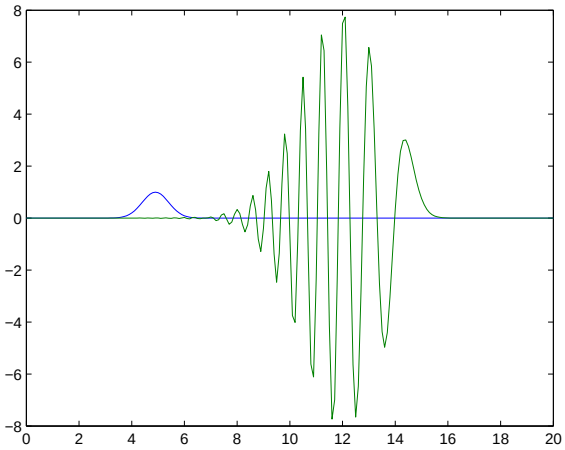


Figure 6.2 Solution of the linear wave equation with $a = 0.4$, using central difference scheme.

$$u(x - \Delta x) = u(x) - \Delta x \frac{\mathrm{d}u}{\mathrm{d}x} + \frac{\Delta x^2}{2} \frac{\mathrm{d}^2 u}{\mathrm{d}x^2} + \ldots, \tag{6.8}$$

and thus

$$\frac{\mathrm{d}u}{\mathrm{d}x} = \frac{u(x) - u(x - \Delta x)}{\Delta x} + \frac{\Delta x}{2} \frac{\mathrm{d}^2 u}{\mathrm{d}x^2} + \ldots \tag{6.9}$$

With this discretization, we are actually solving the differential equation

$$\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} - a \frac{\Delta x}{2} \frac{\partial^2 u}{\partial x^2} + \ldots = 0. \tag{6.10}$$

We recognize the third term represents diffusion. Indeed, we observe from the numerical solution in **Figure 6.1** that the first-order up-wind scheme cause the broardening of the wave packet. This type of discretization error is called numerical diffusion, and is a common problem in PDEs with convection terms. ⋆

If we examine the central difference scheme and the second-order up-wind scheme, we realize that both schemes attempts to remedy the numerical diffusion. However, there is still higher order error in the finite-difference approximation. To further understand the physical implication of the discretization error, we do a Fourier analysis.

6.3 Fourier Error Analysis

The exact first derivative of $\exp(\mathrm{j}\kappa x)$ is

$$\frac{\partial \mathrm{e}^{\mathrm{j}\kappa x}}{\partial x} = \mathrm{j}\kappa \mathrm{e}^{\mathrm{j}\kappa x}. \tag{6.11}$$

On the other hand, using the central difference discretization, the approximated first derivative becomes

⋆ Drift current is a convection term.

$$
\begin{aligned}
(\delta_x u)_j &= \frac{u_{j+1} - u_{j-1}}{2\Delta x} \\
&= \frac{\left(\mathrm{e}^{\mathrm{j}\kappa\Delta x} - \mathrm{e}^{-\mathrm{j}\kappa\Delta x}\right)\mathrm{e}^{\mathrm{j}\kappa x}}{2\Delta x} \\
&= \mathrm{j}\frac{\sin\kappa\Delta x}{\Delta x}\mathrm{e}^{\mathrm{j}\kappa x} \\
&= \mathrm{j}\kappa^*\mathrm{e}^{\mathrm{j}\kappa x}
\end{aligned} \tag{6.12}
$$

where $\kappa^* = \frac{\sin\kappa\Delta x}{\Delta x}$.

Following the same procedure in the beginning of this chapter, we use this approximated derivative to solve the PDE analytically. Now the wavenumber κ in ODE **(6.3)** must be substituted by κ^*.

$$
u_\kappa(x,t) = C_\kappa \mathrm{e}^{\mathrm{j}\kappa(x-a^*t)}, \tag{6.13}
$$

where

$$
a^* = a\frac{\kappa^*}{\kappa} = a\frac{\sin\kappa\Delta x}{\kappa\Delta x} \tag{6.14}
$$

Apparently the phase velocity $c = a^*$ now depends on the wavenumber κ. In this case, $a^* \leq a$, so short-wave-length components propagates too slowly. ★

All the three discretization scheme causes dispersion of some kind. It is interesting to note that, for the first-order up-wind scheme, a^* is complex, and the leading error term is imaginary. ★★ This means that over time, the wave packet will decay in amplitude, and spread out in space ★★★. In general, dissipative discretization schemes are more numerically stable.

As $\kappa\Delta x \to 0$, $a^* \to a$, so small grid size is preferred for high fidelity. However, small spatial grid size would require small time-step, otherwise instability will also occur. Considering the computational cost of having both fine grid in space and time, choosing a good ‡ discretization scheme is very important. What is the proper discretization scheme for the Shockley's equations? We shall answer this question in the next chapter.

★ In optics, we call this *dispersion*.

★★ Try to work this out.

★★★ we call this *dissipation*.

‡ (stable and hi-fi)

7 Scharfetter-Gummel Discretization

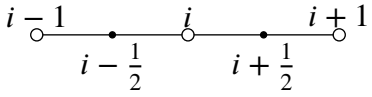


Figure 7.1 Three adjacent nodes $i-1$, i and $i+1$ in a 1D finite volume discretization.

In our naive discretization scheme for the Shockely's equations on a 1D grid (**Figure 7.1**), the electron drift-diffusion current was written as

$$J_{n,i+\frac{1}{2}} = q\mu_n \frac{n_i + n_{i+1}}{2} E + qD_n \frac{n_{i+1} - n_i}{\Delta x}. \tag{7.1}$$

It can be shown that, this is essentially the central difference discretization for both drift and diffusion current. ★

We realize that the drift current is a convection term, and the central difference discretization leads to oscillation. The diffusion term, on the other hand, tend to stablize the discretized equation. Therefore, the stability of the equation is determined by the relative strength of drift and diffusion. In fact, it can be shown that oscillation occurs when ★★

$$\frac{qE}{kT}\frac{\Delta x}{2} \geq 1. \tag{7.2}$$

Therefore, the potential difference between neighboring grid points must be less than half of the thermal potential. This is too stringent a requirement, and is not practical in most devices. A much more stable discretization scheme was proposed by Scharfetter and Gummel in 1969, ★★★ and have been used in most semiconductor devices simulation programs since then. In the following sections, we shall first derive the S-G discretization formula, and follow by a discussion on its physical implications.

7.1 Derivation of S-G Discretization

Examining **(7.1)** more closely, we had made the assumption that the carrier concentration n varies linearly between adjacent nodes. However, we learn from experience that the carrier concentration often varies exponentially in space. Therefore, we hope to have a better approximation to the carrier concentration profile.

Consider the 1D finite volume discretization problem in **Figure 7.1** One obvious heuristic is that the current between adjacent nodes i and $i+1$ will be constant if we ignore generation and recombination. Under this assumption, the electron current equation

$$J_n = q\mu \left(n(x)E + V_T \frac{\mathrm{d}n}{\mathrm{d}x} \right) \tag{7.3}$$

★ write down $J_{n,i-\frac{1}{2}}$ as well, and take the difference of the two.

★★ Consider a PN junction diode with $N_a = 1 \times 10^{19}$ cm^{-3}, $N_d = 1 \times 10^{16}$ cm^{-3}. If we want to solve the Shockley's equations using the central difference scheme, what is the required minimum grid spacing?

★★★ D.L. Scharfetter and H.K. Gummel, ``Large-signal analysis of a silicon Read diode oscillator'', IEEE Transaction on Electron Devices, Vol16, pp.64-77 (1969).

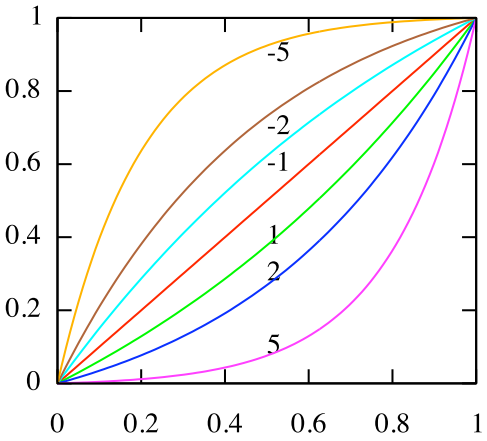


Figure 7.2 Carrier concentration weight function g(x).

can be treated as an ordinary differential equation with J_n as a constant. The boundary conditions are $n|_{x=x_i} = n_i$ and $n|_{x=x_{i+1}} = n_{i+1}$.

The general solution of this ODE is ★

$$n(x) = \frac{V_T}{E} C_0 + C_1 \exp\left(-\frac{E}{V_T} x\right), \tag{7.4}$$

where C_0 and C_1 are constants. Considering the boundary conditions, we can determine C_0 and C_1, and the electron concentration profile between x_i and x_{i+1} is

$$n(x) = [1 - g(x)]n_i + g(x)n_{i+1} \tag{7.5}$$

where

$$g(x) = \frac{1 - \exp\left(\frac{\phi_{i+1}-\phi_i}{V_T} \frac{x-x_i}{\Delta x}\right)}{1 - \exp\left(\frac{\phi_{i+1}-\phi_i}{V_T}\right)}. \tag{7.6}$$

The function $g(x)$ is plotted in **Figure 7.2**, with $\frac{\phi_{i+1}-\phi_i}{V_T}$ as a parameter, which is the potential difference between adjacent grid nodes.

We can observe that, when the potential difference is small, the electron concentration varies almost linearly between the two nodes. On the other hand, when the potential difference is large, the electron concentration deviate strongly from the linear profile. Near the low-potential end, the electron concentration n varies slowly, while near the high-potential end, n varies quickly.

After we obtain the electron concentration $n(x)$, we can derive the electron concentration, gradient of electron concentration, and hence the electron current at the middle of the segment $(i + \frac{1}{2})$

★

$$n|_{i+\frac{1}{2}} = n_i \,\mathrm{aux2}\left(\frac{\phi_i - \phi_{i+1}}{2V_T}\right) + n_{i+1}\,\mathrm{aux1}\left(\frac{\phi_{i+1} - \phi_i}{2V_T}\right) \tag{7.7}$$

$$\frac{\mathrm{d}n}{\mathrm{d}x}|_{i+\frac{1}{2}} = \mathrm{aux1}\left(\frac{\phi_i - \phi_{i+1}}{2V_T}\right)\frac{n_{i+1} - n_i}{\Delta x} \tag{7.8}$$

$$J_n|_{i+\frac{1}{2}} = q\mu\frac{V_T}{\Delta x}\left[n_{i+1}\,\mathrm{B}\left(\frac{\phi_{i+1} - \phi_i}{V_T}\right) - n_i\,\mathrm{B}\left(\frac{\phi_i - \phi_{i+1}}{V_T}\right)\right] \tag{7.9}$$

where we have defined ★

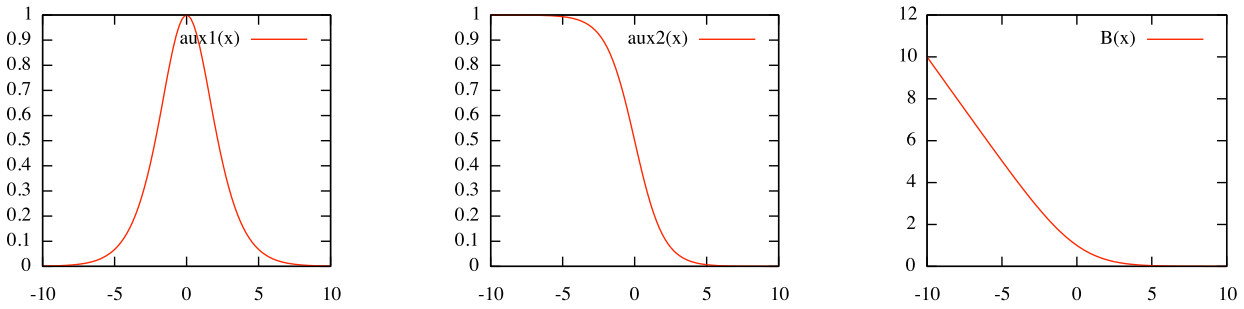
$$\mathrm{aux1}(x) = \frac{x}{\sinh(x)} \tag{7.10}$$

$$\mathrm{aux2}(x) = \frac{1}{1 + e^x} \tag{7.11}$$

$$\mathrm{B}(x) = \frac{x}{e^x - 1} \tag{7.12}$$

.

The three functions are plotted in **Figure 7.3**.



Aux1 function Aux2 function Bern function

Figure 7.3 Auxiliary functions

In practical semiconductor device simulators, we use **(7.9)** in place of **(7.1)** in the finite-volume discretization. However, the meaning of **(7.9)** is not very clear. We attempt to clarify its physical and mathematical implications in the following sections.

7.2 S-G Discretization in Diffusion and Drift limits

If the potential difference between i and $i+1$ is small, the carrier transport is dominated by diffusion. The Bernouli function $\mathrm{B}(x) \approx 1$ as x approaches zero. The half-node current of **(7.9)** thus degenerates to

$$J_n|_{i+\frac{1}{2}} \approx qV_T\mu\frac{n_{i+1} - n_i}{\Delta x}, \tag{7.13}$$

which is simply the diffusion current term in central difference discretization.

★ called the auxiliary 1, auxiliary 2 and Bernouli functions.

On the other hand, assuming $\phi_{i+1} \gg \phi_i$, the high E-field makes drift the dominant transport mechanism. The electron flow by drift from i to $i+1$. In this situation, we have

$$\mathrm{B}\left(\frac{\phi_{i+1}-\phi_i}{V_T}\right) \approx 0$$

and

$$\mathrm{B}\left(\frac{\phi_i-\phi_{i+1}}{V_T}\right) \approx \phi_{i+1}-\phi_i,$$

The electron current $J_n|_{i+\frac{1}{2}}$ is

$$J_n|_{i+\frac{1}{2}} = q\mu n_i \frac{\phi_i-\phi_{i+1}}{\Delta x}, \tag{7.14}$$

which we recognize as the drift current. Note that in this drift current expression, we are using the electron concentration at the up-stream node i, and discarded the down-stream electron concentration totally.

In reality, the E-field in the device is some where between the two extreme cases, and both diffusion and drift exist. It can be shown that in general, the S-G discretization favors using the up-stream concentration information for drift current calculation. The relative contribution of n_i and n_{i+1} to the drift current actually is determined by the interpolation function $g(x)$ plotted in **Figure 7.2**. As E-field increases, it is apparent that $g(0.5)$ in the middle of the segment would give up-stream electron concentration higher weight.

Recall that in the last chapter we have demonstrated that up-wind discretization of convection terms is more stable than the central difference scheme. As S-G scheme is an up-stream scheme, it is not surprising to learn that S-G is more stable than our previous simple central difference scheme.

However, we also learn from the last chapter that up-wind schemes introduces undesirable artificial diffusion. In the following, we shall examine the S-G scheme from this perspective.

7.3 S-G Discretization and Artificial Diffusion

With some arithmetics, the S-G current in **(7.9)** can be written as

$$J_n|_{i+\frac{1}{2}} = q\mu\frac{n_i+n_{i+1}}{2}E + qD\frac{n_{i+1}-n_i}{\Delta x} + qD_a\frac{n_{i+1}-n_i}{\Delta x}, \tag{7.15}$$

where

$$D_a = D\left(\frac{\phi_{i+1}-\phi_i}{2V_T}\coth\frac{\phi_{i+1}-\phi_i}{2V_T} - 1\right) \tag{7.16}$$

We can recognize that the first two terms in **(7.15)** are the drift and diffusion current, discretized in central difference scheme, which is the same as our naive attempt. However, the S-G scheme introduced the third term, which appears like diffusion. Indeed this is called an artificial diffusion term.

The normalized artificial diffusivity D_a is plotted in **Figure 7.4**.

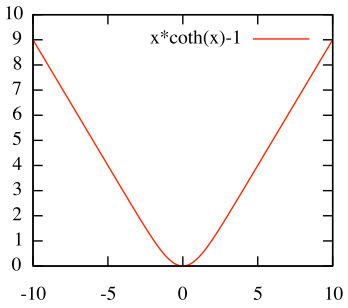


Figure 7.4 Normalized artificial diffusivity Da

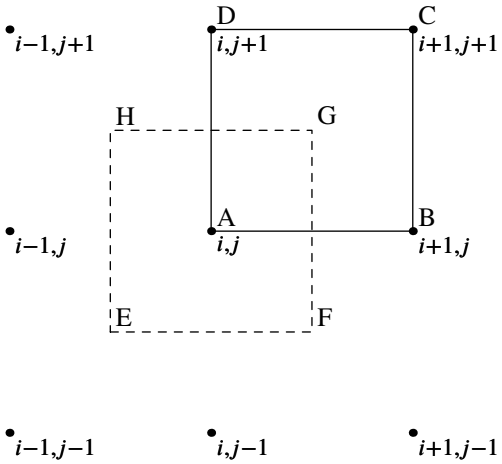
It is obvious that when electric field is small, the artificial diffusion diminishes to zero. When the E-field is high, the S-G scheme introduces some artificial diffusion, so that the drift term is guaranteed to be stable. From this perspective, the S-G discretization can be said to be a central difference scheme plus an adaptive artificial diffusion term.

8 Triangular Mesh for General 2D Geometry

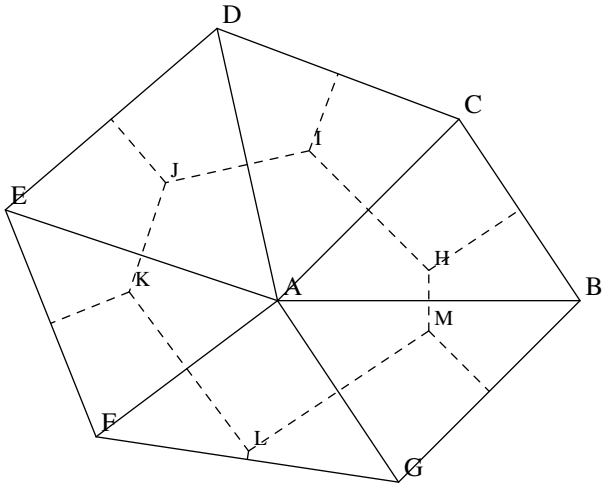
So far, we have been working on either 1D problems or 2D problems on rectangular quadrilateral mesh grids. Each *element* in the structured mesh is a rectangle (e.g. ABCD in **Figure 8.1**a). Each Voronoi *cell* is also a rectangle (e.g. EFGH).

8.1 Triangular Mesh

However, to represent more complex 2D geometry shapes, one prefer to use the triangular mesh elements, as shown in **Figure 8.1**b. ★ To construct the Voronoi cell for vertex A, we must first have a Delaunay triangulation of all the vertices. **Figure 8.1**b shows such a triangulation. we find the perpendicular bisector to each of the edges AB, AC, AD, etc. These bisectors encloses the region HIJKLMN, and is the Voronoi cell for vertex A.



a) Elements (ABCD) and Voronoi cells (EFGH) in a rectangular mesh.



b) Elements (ABC, ACD, ADE, etc) and a Voronoi cells (HIJKLM) in a triangular mesh.

Figure 8.1 Quadrilateral and triangular mesh

★ *Voronoi cell*: For a collection of vertices, each vertex has a Voronoi cell, and for any point within the Voronoi cell, its distance to this vertex is shorter than its distance to any other vertex.
Delaunay triangluation: For any triangle, its circumcircle should not contain any other vertices.

The concept of Voronoi cell and Delaunay triangulation can be extended to 3D as well. For 3D problems, rectangular hexahedral and tetrahedral elements are most commonly used, although other shapes can be used as well.

The construction of Delaunay triangular and tetrahedral mesh is an important field of study in computational geometry, and existing algorithms are pretty time-consuming. Straightforward algorithm to construct a delaunay triangulation takes $O(n^2)$ time for n vertices, while more advanced algorithms takes $O(n \log n)$.

In practice, the user inputs are often given as a collection of polygons, segments and points. The structure may contain several connected sub-domains, e.g. a silicon substrate region and a gate oxide region. A mesh-generation program will determine the locations to insert vertices, and construct a Delaunay mesh. Several constraints must be considered during mesh generation. The most common constraints are

- boundary between sub-domains must be preserved (as triangle edges)in the triangulation;
- maximum area for triangles, and
- minimum angle in triangles ★.

Robust and efficient mesh generation program is difficult to write, and 3D Delaunay mesh generation remains an open research topic.

8.2 Finite-Volume Method and Voronoi Cell

The idea of Voronoi cell is central to the finite-volume method. Recall in the finite-volume discretization of the Shockley equations

$$\oint_S \vec{D} \cdot \overrightarrow{\mathrm{d}s} = \int_V (N_\mathrm{D} - N_\mathrm{A} + p - n)V \tag{8.1}$$

$$\frac{1}{q} \oint_S \vec{J}_n \cdot \overrightarrow{\mathrm{d}s} = \int_V (\frac{\partial n}{\partial t} - G)\,\mathrm{d}V \tag{8.2}$$

$$\frac{1}{q} \oint_S \vec{J}_p \cdot \overrightarrow{\mathrm{d}s} = \int_V (-\frac{\partial p}{\partial t} + G)\,\mathrm{d}V, \tag{8.3}$$

the surface and volume integration for this control volume can be discretized as

$$\oint_S \vec{D} \cdot \overrightarrow{\mathrm{d}s} = \sum_k D_k s_k$$

$$\oint_S \rho\,\mathrm{d}V = \rho v$$

where s_k is the area of the k-th sub-face of the Voronoi cell, and v is the volume of the Voronoi cell.

With reference to the rectangular mesh in **Figure 8.1**a, the control volume for node A is enclosed by the surface EFGH. Let us label the four faces FG, GH, HE and EF with $k = 1 \ldots 4$. For $k = 1$, we evaluate the $\vec{D}_1$ vector by taking the difference of the potential at A and B, and FG is the sub-face s_1.

★ 20.7° guaranteed, 33.8° typical

For the triangular mesh in **Figure 8.1**b, the Voronoi cell of node A has 6 sub-faces. The vector $\vec{D}$ evaluated along AB is multiplied by the area of sub-face HM, and the procedure is repeated on all sub-faces. ⋆

8.3 Assembly of Equations

To avoid repeating some expensive calculations many times, one can assemble the equation in the following way.

- Iterate over all edges to add all flux-related (surface integral) terms in the equation. For example, for edge AB in **Figure 8.1**b, we compute the electron current using S-G formula, which is an expensive operation. This flux term is added to the electron continuity equations for node A, and the same term is added to the equation for node B with the opposite sign.
- Iterate over all cells to add volume integral terms.

  The volume and area of Voronoi cells are all pre-computed, in order to save repeated calculation.

8.4 Exercise

1. Randomly place 10 points on paper. Construct the Delaunay triangulation and its Voronoi graph.
2. Does this triangulation contain obtuse angle? If it does, what happens to the Voronoi cell of that vertex?

⋆ What happens if one angle of the triangle is greater than 90°?

9 Boundary Conditions

In Mathematics, the most common types of boundary conditions of PDE are Dirichlet and Neumann boundary conditions. On the other hand, in semiconductor devices, we categorize the boundaries and interfaces according to the physical phenomena involved at the boundary or interface. In this chapter, we shall discuss how we describe the physics at the boundaries with appropriate boundary conditions, and further discretize the boundary equations on mesh grids.

- Outer boundary of the device structure
- Ohmic contact
- Schottky contact
- Gate contact
- Semiconductor-insulator interface

9.1 Outer boundary of the device structure

9.1.1 Natural boundary

Consider a grid node i,j lying on the outer boundary of the device. As usual we follow the finite-volume method, and assign a cell (ABCD) to the node (shaded region in **Figure 9.1**). In this case, one edge of the cell AD is on the boundary. The volume of this boundary cell is $V = \frac{1}{2}\Delta x \cdot \Delta y$, which is half of the volume of a inner cell.

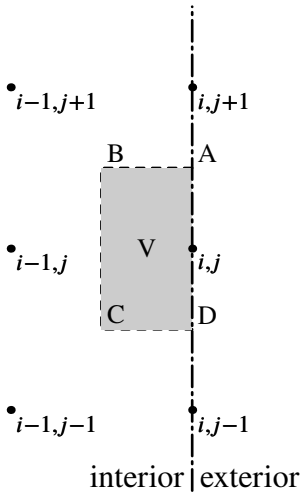


Figure 9.1 Mesh nodes along the outer boundary of the structure.

The most commonly used assumptions at the outer boundary are

- Zero E-field along the boundary surface normal;
- Zero carrier flow along the boundary surface normal.

One recognizes that under these assumptions the boundary conditions to the Poisson and continuity equations are the Neumann boundary conditions $\frac{\partial u}{\partial n} = f$, with $f = 0$. This is often referred to as the *natural boundary condition*.

For the above assumptions to hold, the boundary must be far away from the active region of the device.

If the natural boundary surface is flat, and extend through the entire device structure, the above assumptions imply that the electrostatic potential and carrier concentration profiles are symmetric about the boundary surface. ★ Consider the device structure in **Figure 9.2**, if we solve the PDEs in the shaded region, and assign the natural boundary condition along the dash-dotted line. The solution would be exactly the same as if we solve the equations in the entire structure, including the region enclosed in the dashed line. The solution would be symmetric about the dash-dotted line.

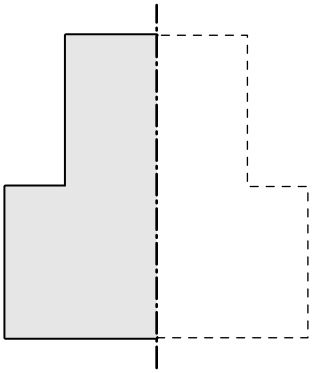


Figure 9.2 Natural boundary condition as a symmetry axis.

According to the assumptions, we can write the followings along the boundary

$$\vec{E} \cdot \hat{n} = \frac{\partial \phi}{\partial n} = 0 \tag{9.1}$$

$$\overrightarrow{J_n} \cdot \hat{n} = 0 \tag{9.2}$$

$$\overrightarrow{J_p} \cdot \hat{n} = 0, \tag{9.3}$$

where $\hat{n}$ is the unit normal vector of the surface boundary.

Consider the boundary cell in **Figure 9.1**, and we evaluate the surface integrals on the peripheral ABCD and volume integrals in the volume V. According to the assumption of the natural boundary condition **(9.1)** - **(9.3)**, the surface integrals for the face AD are all zero. If we follow the equation assembly process outlined in the previous chapter, there is not an edge that provides a flux term to the AD sub-face. Therefore, the assembly process does not need any modification for the natural boundary conditions. ★★

Note that we can not set all boundaries of a device to the natural boundary condition, otherwise the equations are indefinite.

9.1.2 Ohmic contacts

Ohmic contacts occur between a metallic electrode on the boundary and a semiconductor region.

The basic assumptions about ohmic contact is that the recombination rate at ohmic contact is infinity. As a result, carriers concentration at ohmic contacts are the thermal equilibrium values.

★ Recall the method of images in electrostatics. In principle, the surface does not have to be flat either, though the math will be more difficult in those cases.

★★ This is why we call it natural.

The electron and hole concentrations at the ohmic boundary are thus

$$n = \frac{N_D - N_A + \sqrt{(N_D - N_A)^2 + 4n_i^2}}{2} \tag{9.4}$$

$$p = \frac{N_A - N_D + \sqrt{(N_D - N_A)^2 + 4n_i^2}}{2} \tag{9.5}$$

,

where N_D and N_A are the donor and acceptor concentrations, and n_i is the intrinsic carrier concentration in semiconductor.

Since the ohmic contact involves two materials (metal and semiconductor),

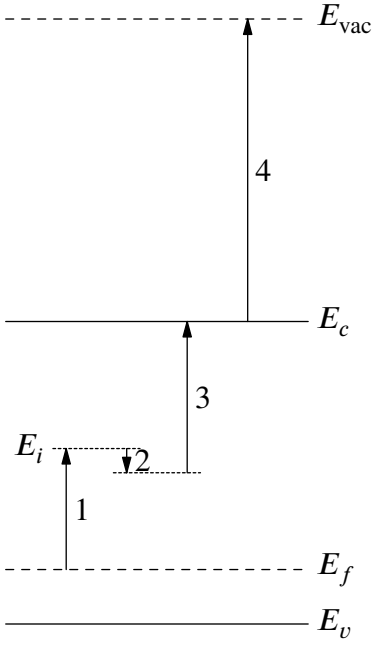


Figure 9.3 Band diagram at an ohmic boundary.

we need a voltage reference that is convenient for both materials. A common choice is to use the applied voltage on the terminal as the zero-potential reference, and use the vacuum potential level as the electrostatic potential in the Poisson's equation.

The electron and hole quasi fermi levels must coincide with the fermi level in metal at the ohmic boundary due to the infinite recombination rate. We thus have

$$\phi_n = \phi_p = -\frac{E_f}{q}, \tag{9.6}$$

and the intrinsic potential in semiconductor

$$\begin{aligned} \phi_i &= \phi_n + \frac{kT}{q} \log\left(\frac{n}{n_i}\right) \\ &= \phi_p - \frac{kT}{q} \log\left(\frac{p}{n_i}\right) \end{aligned}$$

$$= V_{\text{app}} + \frac{kT}{q} \operatorname{asinh}\left(\frac{N_D - N_A}{2n_i}\right),$$

where V_{app} is the voltage applied on the electrode (fermi level in metal).

Referring to the band diagram in **Figure 9.3**, the vacuum potential ϕ can be written as

$$\phi = V_{\text{app}} + \underbrace{\frac{kT}{q} \operatorname{asinh}\left(\frac{N_D - N_A}{2n_i}\right)}_{1} - \underbrace{\frac{kT}{2q} \log\left(\frac{N_c}{N_v}\right)}_{2} - \underbrace{\frac{E_g}{2q}}_{3} - \underbrace{\frac{\chi}{q}}_{4}. \tag{9.7}$$

Equation **(9.7)**, **(9.4)** and **(9.5)** are the equations for the ohmic boundary node i,j in **Figure 9.1**. Obviously these three equations are the simple Dirichlet boundary conditions.

9.1.3 Gate contacts

Gate contacts occurs between a metallic electrode on the boundary and an insulation region.

There is only one unknown (potential) on each grid node in insulator region. We thus need only one equation for the boundary condition:

$$\phi = V_{\text{app}} - \phi_M, \tag{9.8}$$

where ϕ is the vacuum electrostatic potential, V_{app} is the applied voltage (fermi level in metal), and ϕ_M is the work-function of the metal.

Similar to the case of ohmic contact, the equation at a gate contact form a Dirichlet boundary condition.

9.2 Semiconductor-Insulator Interface

The semiconductor-insulator interface occurs all too common in devices. In MOSFETs as well as many other devices, this is the most critical interface to the device operation.

Consider the interface depicted in **Figure 9.4**, and assume we have semiconductor on the left-hand side and insulator on the right-hand side.

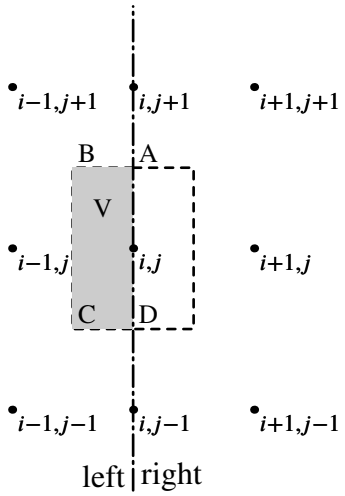
The physics at the interface requires that

- No carrier flows through the interface, or

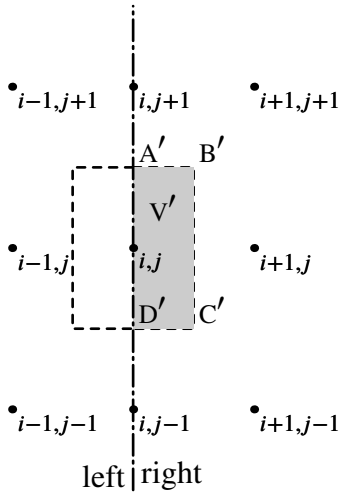
$$\vec{J_n} \cdot \hat{n} = 0$$
$$\vec{J_p} \cdot \hat{n} = 0.$$

- The usual boundary condition between two dielectric layers. This means, in the normal and tangential direction, we have

$$\varepsilon_s \left.\frac{\partial \phi}{\partial n}\right|_{\text{left}} - \varepsilon_i \left.\frac{\partial \phi}{\partial n}\right|_{\text{right}} = \sigma$$
$$\left.\frac{\partial \phi}{\partial t}\right|_{\text{left}} - \left.\frac{\partial \phi}{\partial t}\right|_{\text{right}} = 0,$$



a) Semiconductor side



b) Insulator side

Figure 9.4
Semiconductor-insulator interface.

where ε_s and ε_i are the permittivity in the semiconductor and insulator regions, respectively, σ is the interface charge density (cm^{-2}).

There are several ways to discretize this interface, we shall introduce a scheme that is most flexible in complex geometries.

We first look at the semiconductor side, the node i,j has an control volume V (shaded region in **Figure 9.4**a). There are three unknowns ϕ, n and p for each node in this region. The control volume has three neighbors in this semiconductor region, $(i,j+1)$, $(i-1,j)$ and $(i,j-1)$. Electric displacement and carrier flux are can be calculated only in these three directions. The flux through sub-face AD requires information on the other side of the interface, which we do not to know. ★

According to the first assumption above, the boundary equations for electron and hole concentration are the same as in the case of natural boundaries. On the other hand, for the Poisson's equation

$$\sum_k D_k s_k + F_l = \rho V, \tag{9.9}$$

★ pretend we don't know.

there is some displacement flux F_l through the sub-face AD.

Now let us turn to the insulator side (**Figure 9.4**b). There is only one unknown ϕ' for each node in this region. The control volume V' for node i,j has three neighbors in the insulator region. The Poisson's equation for node i,j is

$$\sum_{k'} D_{k'} s_{k'} + F_r = 0, \tag{9.10}$$

where F_r is the flux through sub-face A′D′.

According to the second assumption for the interface, we have

$$F_l + F_r + \sigma S_{\mathrm{AD}} = 0 \tag{9.11}$$

where σ is the interface charge density, and S_{AD} is the area of the sub-face.

Therefore, **(9.9)** and **(9.10)** can be combined as

$$\sum_{k} D_k s_k + \sum_{k'} D_{k'} s_{k'} = \rho V + \sigma S_{\mathrm{AD}}. \tag{9.12}$$

Certainly, we expect the potential to be continuous across the boundary, therefore

$$\phi = \phi'. \tag{9.13}$$

9.3 Exercise: 1D Shockley equations (PN junction diode) ★

Consider a PN-junction diode with base length of 10 μm on each side. Doping concentrations are $N_A = 10^{19}$ cm^{-3} and $N_D = 10^{16}$ cm^{-3}. Assume that the minority carrier life-time is 10^{-7} s. Solve the Shockley's equation when the diode is 1) in equilibrium, and 2) forward-biased at 0.5 V. Plot the potential, and carrier concentration profile in the device.

★ 20 Programming Credits

10 Automatic Differentation

In **Chapter 4**, we have seen that solving the of non-linear equations

$$\boldsymbol{F}(\boldsymbol{v}) = \boldsymbol{0}.$$

requires us to compute the Jacobian matrix

$$\boldsymbol{J} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{pmatrix}.$$

For the simple example in **Chapter 4**, we derived the analytic expressions for each element in the Jacobian matrix. However, the Shockley's equations in **Chapter 3** and the Scharfetter-Gummel discretizatioin equations in **Chapter 7** are much more complicated. Manually compute the partial derivatives is painful and prone to error. To make it worse, practical device simulators supports dozens of carrier mobility models, and supports many different mesh elements, which multiplies the complexity of Jacobian calculation.

There are three alternative approaches to calculate the partial derivatives with computer:

- Symoblic differentiation. Symbolic mathematics engines ``understands'' the expression, ``knowns'' the rules of differentiation, and derives the derivatives analytically as a human does it. Popular symbolic software include `Mathematica` and `Maple`. Symbolic differentiation is slow, and the software is very complex.

- Numerical differentiation. We use the forward difference to approximate the derivative

  $$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h},$$

  if h is sufficiently small. Numerical differentiation is slow if we need derivatives against many independent variables. Additionally, accuracy of numerical differentiation is poor due to round-off errors.

- Automatic differentiation. Our topic in this chapter. Exact derivatives, easy to implement, and fast.

  Consider the function

  $$f(x_1,x_2) = \sin(x_1) + x_1 x_2, \tag{10.1}$$

where x_1 and x_2 are the two independent variables. We want to evaluate the partial derivatives with respect to x_1 and x_2. More specifically, as an example, for $x_1 = \frac{\pi}{4}$, $x_2 = 4$, we want to calculate $f(x_1,x_2)$ and $\frac{\partial f(x_1,x_2)}{\partial x_1}$ and $\frac{\partial f(x_1,x_2)}{\partial x_2}$. With automatic differentiation, we can compute the derivatives with the following code:

```
x1 = ADVar(3.14159265/4.0, 0)    # AD Variable with index 0
x2 = ADVar(4.0, 1)               # AD Variable with index 1
```

```
y = sin(x1) + x1*x2
print y

### output:
# value:3.84869943055
# deriv: [(0, 4.707106781821139), (1, 0.78539816250000005)]
```

which agrees with our hand-calculation of

$$\frac{\partial f}{\partial x_1} = \cos(x_1) + x_2 = \frac{\sqrt{2}}{4} + 4$$
$$\frac{\partial f}{\partial x_2} = x_1 = \frac{\pi}{4}$$

In the following sections, we shall demonstrate how the above program calculates the derivatives automatically.

10.1 Computation Graph

The function **(10.1)** can be expressed as a computation graph, as shown in **Figure 10.1**. Starting with independent variables x_1, x_2, we calculate the function with 5 intermediate variables $w_1 \ldots w_5$.

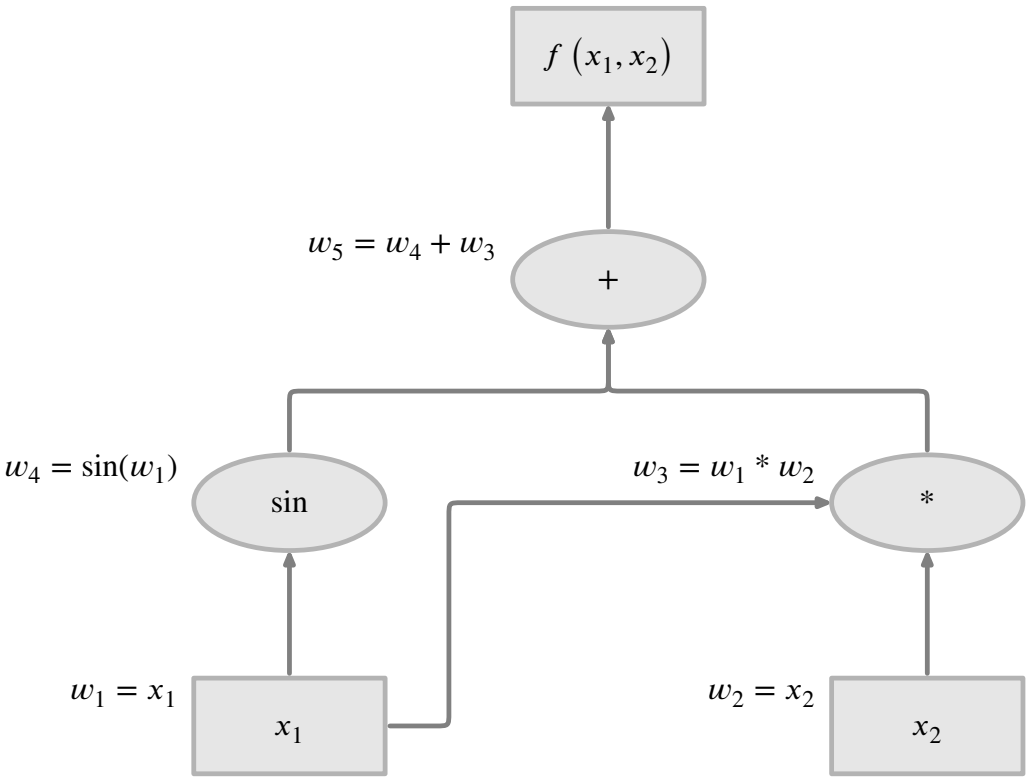


Figure 10.1 Computation graph of the function $f(x,y)$.

10.2 Forward- and Backward-Accumulation

Automatic differentiation calculation can be performed by traversing the computation graph. Either forward- or backward-mode can be used.

Before we proceed, let us refresh ourselves with the rules of differentiation

$$\frac{\partial u(v(x))}{\partial x} = \frac{\partial u}{\partial v}\frac{\partial v}{\partial x} \qquad \text{Chain rule}$$
$$\frac{\partial (u \cdot v)}{\partial x} = v\frac{\partial u}{\partial x} + u\frac{\partial v}{\partial x}$$
$$\frac{\partial (u + v)}{\partial x} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial x}$$

In the forward-accumulation mode, we traverse computation graph from bottom to top, as shown in **Figure 10.2**. To calculate $\frac{\partial f}{\partial x_1}$, we seed the computation with $\dot{w}_1 = 1$, $\dot{w}_2 = 0$. Sweeping each node from bottom to top, we will obtain the derivative pretty straight-forwardly. Similarly, for $\frac{\partial f}{\partial x_1}$, we seed the computation with $\dot{w}_1 = 0$, $\dot{w}_2 = 1$.

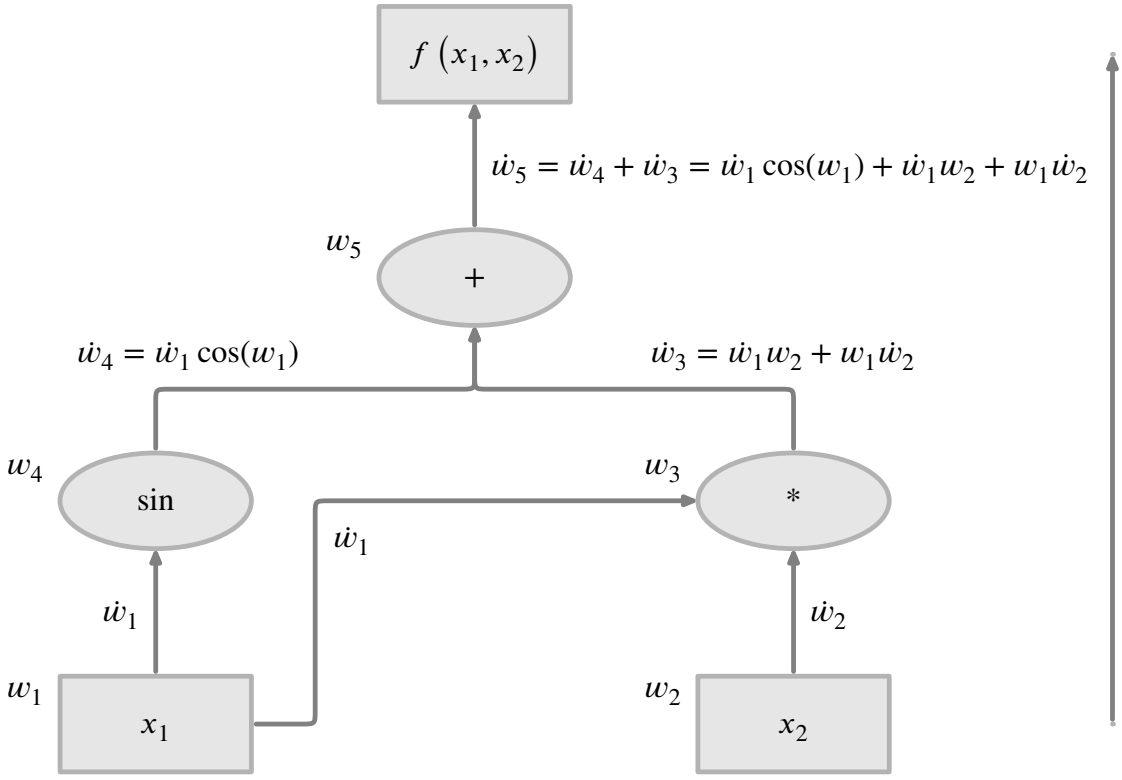


Figure 10.2 Automatic differentiation with forward accumulation.

Alternatively, one can traverse the computation graph from top to bottom (backward accumulation), as shown in **Figure 10.3**.

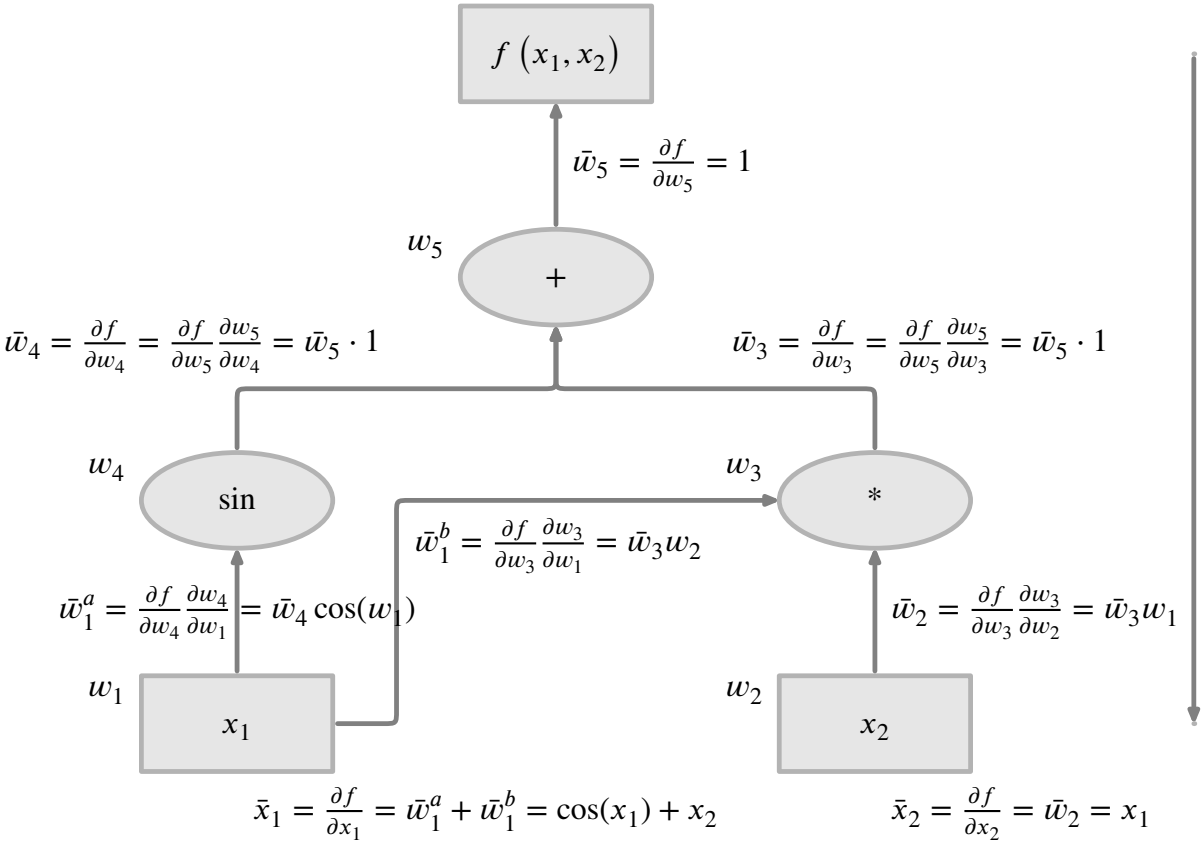

$f(x_1, x_2)$
$\bar{w}_5 = \frac{\partial f}{\partial w_5} = 1$
w_5
+
$\bar{w}_4 = \frac{\partial f}{\partial w_4} = \frac{\partial f}{\partial w_5}\frac{\partial w_5}{\partial w_4} = \bar{w}_5 \cdot 1$
$\bar{w}_3 = \frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial w_5}\frac{\partial w_5}{\partial w_3} = \bar{w}_5 \cdot 1$
w_4
sin
w_3
*
$\bar{w}_1^b = \frac{\partial f}{\partial w_3}\frac{\partial w_3}{\partial w_1} = \bar{w}_3 w_2$
$\bar{w}_1^a = \frac{\partial f}{\partial w_4}\frac{\partial w_4}{\partial w_1} = \bar{w}_4 \cos(w_1)$
$\bar{w}_2 = \frac{\partial f}{\partial w_3}\frac{\partial w_3}{\partial w_2} = \bar{w}_3 w_1$
w_1
x_1
w_2
x_2
$\bar{x}_1 = \frac{\partial f}{\partial x_1} = \bar{w}_1^a + \bar{w}_1^b = \cos(x_1) + x_2$
$\bar{x}_2 = \frac{\partial f}{\partial x_2} = \bar{w}_2 = x_1$

Figure 10.3 Automatic differentiation with backward accumulation.

10.3 Operator Overloading

There are several ways to implement the traversing of the computation graph. The simplest yet widely used approach is based on the operator overloading facility provided in most object-oriented programming languages. ★

We define a class `ADVar` to represent all variables that is involved in the calculation. In the previous example, we create the object `x1` and `x2`:

```
x1 = ADVar(3.14159265/4.0, 0)    # AD Variable with index 0
x2 = ADVar(4.0, 1)               # AD Variable with index 1
```

For the expressions `x1+x2` or `x1*x2` to work, we need to define the arithmetic operators for the `ADVar` class. The following Python code illustrates a minimalist implementation.

```
class ADVar(object):
  def __init__(self, value=0.0, index=-1):
    self.value  = value    # value of this variable
    self.dv_dx1 = 0.0      # partial derivative against x1
    self.dv_dx2 = 0.0      # partial derivative against x2
    if index==0:
      self.dv_dx1 = 1.0
    elif index==1:
      self.dv_dx2 = 1.0
```

★ The alternative to ``operator overloading'' is ``code transformation''.

```
  def __sum__(self, other):      # overloading the + operator
    r = ADVar()
    r.value = self.value + other.value
    r.dv_dx1 = self.dv_dx1 + other.dv_dx1
    r.dv_dx2 = self.dv_dx2 + other.dv_dx2
    return r

  def __mul__(self, other):      # overloading the * operator
    r = ADVar()
    r.value = self.value * other.value
    r.dv_dx1 = self.value*other.dv_dx1 + other.value*self.dv_dx1
    r.dv_dx2 = self.value*other.dv_dx2 + other.value*self.dv_dx2
```

In Python, the following two expressions are equivalent.

```
x1+x2
x1.__sum__(x2)
```

It is easy to see that the above operator-overloading implementation is based on the forward-accumulation mode.

The pyEDA package contains an implementation of automatic differentiation following the algorithms outlined above, but is more general and complete. Similar implementations exist for several popular programming languages. For users of C++, one may look at the ADOL-C package.

10.4 Further Readings

We have examined the procedure of calculating the first derivative of functions with automatic differentiation. It is possible to calculate higher order derivatives as well. For details, readers are referred to the manual of the ADOL-C package. ★

Often one encounters implicitly-defined functions like $f(y,x_1,x_2) = 0$, where y is the dependent variable and x_1, x_2 are the independent variables. For example, the current-voltage relation of a P-N diode can be written as $I_d = AJ_0 \exp\left(\frac{V-I_dR}{nV_T}\right)$, which is an non-linear implicit relation. One hopes to obtain the partial derivatives $\frac{\partial y}{\partial x_1}$ and $\frac{\partial y}{\partial x_2}$. This is possible as well with automatic differentiation, and is implemented in pyEDA.

★ https://projects.coin-or.org/ADOL-C/browser/stable/2.1/ADOL-C/doc/adolc-manual.pdf?format=raw

11 Circuit Simulation

Circuit simulation is one of the core component among EDA software. We are all familiar with the electronic circuit simulator SPICE, first developed by researchers at UC Berkeley starting in 1970s. The internal design of SPICE is very elegant even by today's standard. It is modular, efficient and extensible, allowing further development by the commercial vendors for over two decades.

After 30 years, writing a circuit simulator has become much simpler with the many new software technologies. We shall outline a simple circuit simulator in this chapter. Traditionally, a circuit simulation course starts with linear circuit elements and linear equations. Since we are already familiar with the concept and techniques for non-linear equations, we proceed directly to the non-linear circuit equations.

11.1 KCL and Nodal Analysis

The fundamental physical laws governing the electronic circuits are the Kirchoff's current law (KCL) and voltage law (KVL). ★

By KVL, the net voltage drop along any loop in the circuit is zero. By KCL, the algebraic sum of all the currents flowing into any circuit node is zero.

In circuit simulation, the KCL, and the associated *nodal analysis* is more convenient. In nodal analysis, we assign one unknown to each circuit node, which is the nodal voltage; and KCL provides one equation at that node.

Consider the circuit diagram in **Figure 11.1**,

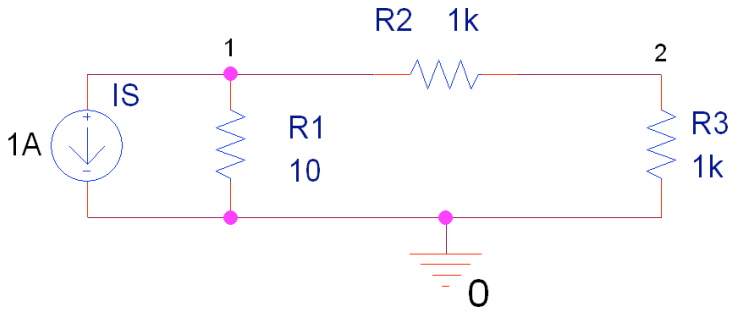


Figure 11.1 A linear circuit.

we have three circuit nodes 0, 1 and 2.

For node 1, the unknown variable is the voltage at node 1, and according to KCL, we have

$$-IS + (V_0 - V_1)/R_1 + (V_2 - V_1)/R_2 = 0 \tag{11.1}$$

Similarly, for node 2, we have

$$(V_1 - V_2)/R_2 + (V_0 - V_2)/R_3 = 0 \tag{11.2}$$

Finally, for node 3,

$$V_0 = 0 \tag{11.3}$$

★ Of course, Maxwell's equations are more fundamental. Both KCL and KVL can be derived from Maxwell's equations.

Therefore the circuit equations (11.1)–(11.3) can be written as $\boldsymbol{F}(\boldsymbol{v}) = \boldsymbol{0}$, and we can use the familiar Newton's method to solve the equations easily.

In practice, it is much easier for a computer program to assemble the equations by circuit elements. Consider the resistor R_2, it contributes currents to the nodes 1 and 2

$$I_{1,R_2} = (V_2 - V_1)/R_2 \tag{11.4}$$

$$I_{2,R_2} = (V_1 - V_2)/R_2 = -I_{R_2,1}. \tag{11.5}$$

We add the branch currents to the corresponding node equations. When all circuit elements are added to the equations in this manner, we write the equation for circuit node i as

$$\sum_{e \in E} I_{i,e} = 0 \tag{11.6}$$

where E is the set of all circuit elements. By this procedure, we shall arrive at the same set of equations of (11.1) – (11.3) .

11.1.1 Independent Voltage Source and Auxiliary Variable

Voltage sources require some special treatment in the nodal analysis. Consider an independent voltage source with its positive terminal connected to the node i, and negative terminal connected to j. We introduce an auxiliary variable I_{src}, which is the current flowing into the negative terminal, through the source, and out of the positive terminal. The contribution of the voltage source to the node equation i and j can be written as

$$I_{i,\text{src}} = I_{\text{src}} \tag{11.7}$$

$$I_{j,\text{src}} = -I_{\text{src}}, \tag{11.8}$$

Since we introduced an auxiliary variable, there is also an auxiliary equation

$$V_i - V_j - V_{\text{src}} = 0, \tag{11.9}$$

where V_{src} is the source voltage.

11.2 Circuit Elements

11.2.1 Resistor

For a resistor R between node i an node j, its contributions is the equations are

$$I_{i,R} = (V_j - V_i)/R \tag{11.10}$$

$$I_{j,R} = (V_i - V_j)/R. \tag{11.11}$$

11.2.2 Capacitor

The current through a linear capacitor C can be written as

$$i = C\frac{\mathrm{d}V}{\mathrm{d}t}. \tag{11.12}$$

We need to approximate the time-derivative with a discretized formula. Here we use a first-order formula, and write the current at the $(n+1)$-th time step as

$$I_{i,C}^{(n+1)} = C\frac{(V_j^{(n+1)} - V_i^{(n+1)}) - (V_j^{(n)} - V_i^{(n)})}{\Delta t} \tag{11.13}$$

$$I_{j,C}^{(n+1)} = C\frac{(V_i^{(n+1)} - V_j^{(n+1)}) - (V_i^{(n)} - V_j^{(n)})}{\Delta t} \tag{11.14}$$

where Δt is the time step.

11.2.3 Inductor

The equation for an inductor is

$$v = L\frac{\mathrm{d}i}{\mathrm{d}t}. \tag{11.15}$$

As with the voltage source, we need to introduce an auxiliary variable I_L, which is the current through the inductor.

We write its contribution to nodal equations, and the auxiliary equation as

$$I_{i,L}^{(n+1)} = I_L^{(n+1)} \tag{11.16}$$

$$I_{j,L}^{(n+1)} = -I_L^{(n+1)} \tag{11.17}$$

$$V_j^{(n+1)} - V_i^{(n+1)} = L\frac{I_L^{(n+1)} - I_L^{(n)}}{\Delta t} \tag{11.18}$$

11.2.4 Diode

Diode is an nonlinear circuit element as oppose to the linear elements we have discussed above. A simplest model for the diode would be the DC current model

$$I = AJ_0\left[\exp(V/V_T) - 1\right] \tag{11.19}$$

where A is the area and J_0 is the saturation current.

However, we can write its contribution to the circuit equations

$$I_{i,D} = AJ_0\left[\exp\left(\frac{V_j - V_i}{V_T}\right) - 1\right] \tag{11.20}$$

$$I_{j,D} = AJ_0\left[\exp\left(\frac{V_i - V_j}{V_T}\right) - 1\right]. \tag{11.21}$$

This simple model does not include the parasitic capacitance (depletion as well as diffusion capacitance), while a practical model (as in SPICE) certainly incorporates these and other parasitic effects.

11.2.5 MOSFET

MOSFET is a four-terminal nonlinear device, and is a lot more complicated than the other devices. However, we still can express the terminal currents in terms of terminal voltages. One has to be careful that the net current flowing in/out of a device through all terminals must be zero, this must be hold at any time instance.

There are too many different MOSFET models for circuit simulation than we can even list here. The most popular family of models are the BSIM models developed at UC Berkeley. The BSIM model starts from a model for the threshold voltage, then a unified steady-state I-V model. The parasitic capacitance model is then added on top of the DC model. Finally, other effects, such as non-quasi static model, noise model, etc. are added. The subject of compact modelling of MOSFETs worths a separate full-semester course, and certainly can not be covered here.

Instead, we list a few useful references for interested readers

- "SPICE 3 User's Manual", available online
- N. Arora, "MOSFET Modeling for VLSI Simulation", World Scientific (2007)
- Weidong Liu, et al, "BSIM3v3 MOSFET Model Users' Manual", available online (1999)

11.2.6 Others

Apart from the devices described above, many other types of semiconductor transistors, transmission lines, op-amps, and superconductor devices have been incorporated into circuit simulators.

11.3 Summary

It is possible to couple circuit simulation based on KCL and device simulation based on Shockley's equation. In the semiconductor simulator, we need to integrate the current density at electrodes to obtain the total terminal current, and connect them to KCL.

Apart from DC and transient simulation we described here, standard circuit simulators can also perform sensitivity analysis, small signal analysis and transfer function analysis. We are unable to cover these topics here.

Another topic that we have not treated formally is the discretization in time. We have been using a straightforward first-order scheme[⋆], which fortunately is very stable. We shall treat more formally, look at the stability and accuracy of the time discretization schemes.

11.4 Exercise

Based on the provided skeleton code for a circuit simulator, write a program to simulate a rectifier circuit. The input is a sine wave with amplitude of 24 V and frequency of 50 Hz. A diode is used to block the negative half of the waveform and conduct the positive half.

⋆ the Backward Euler scheme

12 Floating-Point Rounding Error

We all know that with-in computers, numbers are stored in binary numeral systems. More specifically, integers are stored in the two's complements system illustrated in **Table 12.1**.

binary	decimal	binary	decimal
0111	7	1111	-1
0110	6	1110	-2
0101	5	1101	-3
0100	4	1100	-4
0011	3	1011	-5
0010	2	1010	-6
0001	1	1001	-7
0000	0	1000	-8

Table 12.1 Four-bit signed integers, two's complement system

Digital computers can not handle real numbers directly. Internally, computers uses integers to represent real numbers with finite precision. We shall first look at the format of this internal representation.

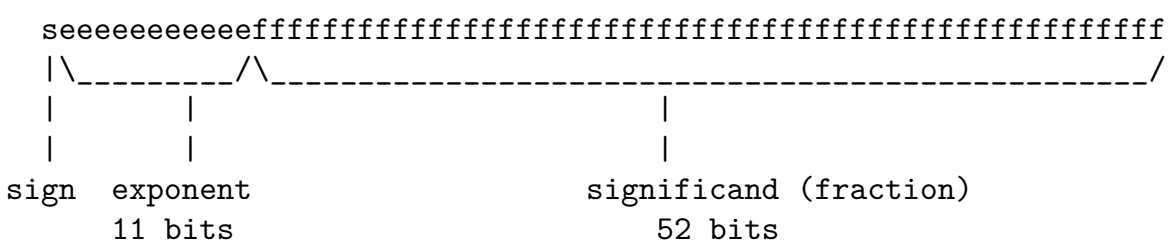
12.1 IEEE 754 Standard for Floating-Point Arithmetics

How to convert the binary number 1.011 to decimal?

$$10.011_2 = (1 \times 2^1 + 1 \times 2^{-2} + 1 \times 2^{-3})_{10} = 2.375_{10}$$

Can we represent decimal number 0.1 exactly in floating point representation? The answer is No!

The most widely used standard for floating-point computation is the IEEE 754 standard. ★ Take the double-precision number as an example, the binary format is



To convert a binary number in double-precision format to decimal, one can use the formula

$$(-1)^{\text{signbit}} \times 2^{\text{exponent}_2 - 1023} \times 1.\text{significandbits}_2$$

Some examples of double-precision floating point numbers are listed below.

★ 1. David Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic". ACM Computing Surveys 23, pp.5-48 (1991)

2. "Numerical Recipes, the Art of Scientific Computing", available online: www.nr.com

```
double format (hex)      value in decimals
3ff0 0000 0000 0000   = 1
3ff0 0000 0000 0001   = 1.0000000000000002, the next higher number > 1
3ff0 0000 0000 0002   = 1.0000000000000004
4000 0000 0000 0000   = 2
c000 0000 0000 0000   = -2
4003 0000 0000 0000   = 2.375
```

From the examples, we learn that the precision of double numbers is about 16 decimal places. Double precision is the most widely used format in scientific computation. However, there are other formats available in the hardware, and used in other applications, as listed in **Table 12.2**.

size	name	C type	significand	Emin	Emax
32	Single precision	float	23+1	-126	127
64	Double precision	double	52+1	-1022	1023
80	Extended precision (x86)	long double	64+1	-16382	16383
128	Quadruple precision	long double	112+1	-16382	16383

Table 12.2 Common floating point types used in computers. Sizes are in bits, Emin and Emax are exponents of base 2.

12.2 Examples of Rounding Error

The 16-digit precision may seem very good compared to manual calculations, but in many situations, this finite precision still causes problem. For example, consider the quadratic equation

$$ax^2 + bx + c = 0, \tag{12.1}$$

the solutions are well known to be

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{12.2}$$

However, calculating from **(12.2)** directly is not the best thing to do. ★ When $b^2 \gg ac$, we have $\sqrt{b^2 - 4ac} \approx |b|$. When we then do the subtraction, the slight difference between two large numbers is lost due to finite precision. This is called a catastrophic cancellation, and is the most common round-off error.

A better algorithm (taken from Numerical Recipe) would be to first calculate

$$q = -\frac{1}{2}\left[b + \mathrm{sgn}(b)\sqrt{b^2 - 4ac}\right] \tag{12.3}$$

then the two roots are

$$x_1 = \frac{q}{a} \quad x_2 = \frac{c}{q}.$$

★ For $a = c = 1$, $b = -10^9$, **(12.2)** yields $x_1 = 1e9$, $x_2 = 0$, while the correct answer should be $x_2 = 1e-9$.

We observe that by carefully rearranging the calculation sequence, we can minimize the rounding off error.

12.3 Function Evaluation to Machine Precision

Consider the Bernoulli function $\mathrm{B}(x) = x/(\mathrm{e}^x - 1)$ used in the S-G discretization. Using this expression directly would lead to 0/0 for $x = 0$. Indeed, for large and small values of x, we should evaluate the function with alternative expressions.

With Taylor expansion, we can avoid catastrophic cancellations, and handle the 0/0 case

$$\mathrm{B}(x) = \frac{x}{\mathrm{e}^x - 1} = \begin{cases} -x & x \le B_0 \\ \dfrac{x}{\exp(x) - 1} & B_0 < x \le B_1 \\ 1 - \dfrac{x}{2}\left[1 - \dfrac{x}{6}\left(1 - \dfrac{x^2}{60}\right)\right] & B_1 < x \le B_2 \\ \dfrac{x\exp(-x)}{1 - \exp(-x)} & B_2 < x \le B_3 \\ x\exp(-x) & B_3 < x \le B_4 \\ 0 & \text{otherwise} \end{cases}$$

for breakpoints $B_0 \ldots B_4$. With proper choice of the numerical values for the breakpoints, The Bernoulli function can be evaluated to machine precision. For machines compatible with the IEEE 754 double precision system, the values are found to be

Breakpoint	Value
B_0	-3.742994775023696e+01
B_1	-1.983224379137254e-02
B_2	2.177550998053050e-02
B_3	3.742994775023696e+01
B_4	7.451332191019408e+02

Table 12.3

This breakpoint technique is widely used in optimized mathematics libraries such as boost[★]. The appropriate expressions for many useful functions in semiconductor device simulation can be found in the source code of the Simulation Generation Framework (SGF) by K.M. Kramer. Or you may look at the PySim package and other demo codes provided in the course website.

12.4 High Precision and Adaptive Precision Arithmetics

The techniques described in the previous sections are very useful to minimize the error due to round-off, but they are not always sufficient. In some situations, we do need higher precision in floating point arithmetics.

★ www.boost.org

For example, consider a set of linear equations $Ax = b$, where the matrix A has very large eigenvalues ($>10^{10}$ often occurs in practice). A minute round-off error in the RHS vector b may cause a large error in the solution vector x in this case. This is a well-known problem, and has driven CPU manufacturers to provide higher-precision floating point hardware. For example, 80-bit floating point number is available on x86 platform as the extended precision format (C type bi long double). IBM PowerPC hardware even provides 128-bit quadruple precision arithmetics. However, precision arithmetics is, as you may have expected, slower. Quadruple arithmetics is about 50 % slower than corresponding double arithmetics. In practice, the vast majority of computations are still using double precision.

Some geometry predicates, such as testing whether a point is within the circumcircle of a triangle, ⋆ requires very high precision. These predicates usually relies on the sign of a determinant, and a minute round-off error may reverse the result completely. As a result, even higher precision is needed. Adaptive precision arithmetics has been implemented to handle these predicates efficiently and accurately. ⋆⋆

As a circuit designer or device engineer, one has to be aware of the importance of handling round-off error properly in calculation. In many fields, these techniques have been thoroughly studied, and can be found in reference books.

12.5 Variable Scaling

We are trained to work in the SI units. ⋆ ⋆ ⋆ However, there are other unit systems that are more convenient in certain areas. The SI unit is particularly bad for computers working on semiconductor devices. The voltage is in the order of 1, and the carrier concentration can be as large as 10^{20} cm^{-3}.

As a result, when we form the Jacobian matrix for the Shockley's equation, we will find some matrix elements has huge values, while others very small. As computers have limited computation precision (normally 16 digits), arithmetics between big and small values often leads to floating-point truncation error, and convergence difficulties. To avoid these errors, we can scale the unknowns and matrix elements.

Another common practice, however, is to work with a more convenient unit system in the simulation program. For example, we can set one length unit = 10 nm, one time unit = 1 ps. We convert all user inputs to our internal unit scale before computation, and convert back after obtaining the solution.

This set of units tend to normalize all variables in semiconductor to the order of 1, where the floating point number representation has maximum precision. Common conversions are summarized in **Table 12.4**.

⋆ very common operation in mesh generator

⋆⋆ J.R. Shewchuk. Robust Adaptive Floating-Point Geometric Predicates. Proceedings of the Twelfth Annual Symposium on Computational Geometry. Association for Computing Machinery, May 1996.

⋆ ⋆ ⋆ We didn't follow it exactly, as we are using cm^{-3} for carrier concentrations.

SI unit	scaled unit
cm	10^6
s	10^{12}
V	1.0
C	$1/1.602176462 \times 10^{-19}$
K	1/300
J	C · V
kg	$J \cdot s^2/m^2$
eV	1.0 · V
A	C/s

Table 12.4 Conversion from SI unit to internal unit

13 Linear Solvers and Conditioning

When we solve the linear PDEs, the final step is the solve the matrix equation

$$\boldsymbol{A} \cdot \boldsymbol{x} = b.$$

When we solve nonlinear PDEs with Newton's iterative method, the search direction $\boldsymbol{d}$ is found by solving

$$\boldsymbol{J}[\boldsymbol{x}^{(k)}] \cdot \boldsymbol{d}^{(k)} = \boldsymbol{F}[\boldsymbol{v}^{(k)}].$$

In both cases, we need a linear solver for sparse matrices.

In device simulators, solving the linear system takes roughly half of the total computation time.

13.1 Direct Solvers based on LU Decomposition

The classical method for solving linear system is the *Gaussian elimination* method or the *LU decomposition* method. In practice, one uses *partial pivoting* to avoid numerical instability. The LU decomposition with partial pivoting is regarded as the most robust among all practical linear solvers, and is used as the fall-back when other algorithms fail.

However, there are severe draw-backs of LU decomposition. Firstly the algorithm takes $O(n^3)$ time and $O(n^2)$ memory, which is notoriously slow, and takes far too much memory. Secondly, even if the matrix to solve is sparse, its LU factors are in general full matrices. The large number of *fills* prevents us from taking advantage of the sparsity of the matrix. Thirdly, the LU decomposition algorithm is inherently serial. It is very difficult to improve its efficiency on multi-core or multi-processor computers.

13.2 Iterative Solvers

Alternatively, there are iterative methods to solve the linear system. We describe the simplest Jacobi method as an example. For the matrix equation $Ax = b$, where $A = \{a_{ij}\}$, we start from an initial guess $x^{(0)}$, and use the iteration

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right) \tag{13.1}$$

to find the next approximated solutioin $x^{(k+1)}$. Note that the Jacobi method takes $O(mk)$ time, where m is the number of non-zero elements in the matrix, and k is the number of iterations. The iterative method takes no additional memory space other than the memory used to store the solution vector and (optionally) the non-zero elements in the matrix. One immediately sees that they are very suitable for sparse linear systems that result from finite-difference/finite-volume discretization. Further, many iterative algorithms can be easily parallelized, which is a very attractive property.

Stationary iterative linear solvers:

- Jacobi,
- Gauss-Seidel,
- Successive Over-Relaxation (SOR).

Krylov sub-space iterative linear solvers:

- Conjugate Gradient method (CG),
- Generalized Minimum RESidual method (GMRES),
- Bi-Conjugate Gradient Stablized method (BiCGStab).

The iterative methods became popular after the SOR method was invented in 1950. The Krylov sub-space methods show better convergence properties, and have been widely used in solving PDEs since the 1970s. The mathematical theory of Krylov sub-space methods may appear too abstract, but there is an intuitive tutorial on it, and is suitable for casual readers. ★

There are many software packages supplying iterative solvers for sparse linear system. For example, PETSc is a C library that provides sparse linear and non-linear solvers for parallel computation. Matlab also provides a rich set of routines for sparse matrix and sparse linear solvers.

13.2.1 Condition Number

The major draw-back of the iterative methods is that it requires the matrix to be *well-conditioned*. The rate of convergence can be (roughly) correlated to the *condition number* of the matrix. A rough definition for the condition number κ is

$$\kappa(\boldsymbol{A}) = \left| \frac{\lambda_{\max}(\boldsymbol{A})}{\lambda_{\min}(\boldsymbol{A})} \right| \tag{13.2}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum ★★ eigenvalues of the matrix $\boldsymbol{A}$.

If the condition number is small, the matrix is said to be well-conditioned, otherwise it is *ill-conditioned*. We shall avoid rigorous mathematics, and list a few intuitive results here

1. Singular (not invertible) matrices has infinite condition number.
2. Identity matrix has condition number of 1.
3. Diagonal-dominant matrices are well-conditioned.
4. For an extremely large eigenvalue λ, a small change to the solution vector x will cause a big change in Ax, and hence big change in the error $e = Ax - b$.
5. For an extremely small eigenvalue, two (or more) very different vectors x_1 and x_2 may results in similarly small error vectors, or $\|e_1\| \approx \|e_2\|$.

If condition number is large, the iterative methods can either be slow to converge, or diverge in some cases. In practice, one needs to *precondition* the matrix before the iteration. We look for a matrix P^{-1} such that $P^{-1}A$ has a much smaller condition number than A. ★★★ Instead of $Ax = b$, we then solve

$$P^{-1}Ax = P^{-1}b \tag{13.3}$$

using the iterative methods. Apparently we want P^{-1} to be as sparse as possible to minimize the computation cost. With preconditioning, the iterative methods converge much faster and are more robust.

★ J.R. Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain", available online (1994)

★★ recall that if $Av = \lambda v$, λ is called an *eigenvalue* and v is the corresponding *eigenvector*.

★★★ we want $P^{-1}A \approx I$.

Therefore, finding a suitable precondition matrix P^{-1} is the key to successfully use the iterative linear solvers. There are many preconditioning algorithms. The Hypre library is a collection of such preconditioners. Among them, the Incomplete LU factorization (ILU) method is a very robust preconditioner for semiconductor device simulation problems. However, the parallelized implementation ILU method are to-date not very efficient.

13.3 Direct Solvers Based on Multi-Frontal Method

The standard LU decomposition method procedure is not compatible with parallel computation. However, careful analysis on the column elimination dependency shows that, for sparse matrices, it is possible to break the decomposition tasks into many independent sub-tasks.

This observation leads to the invention of the multi-frontal method for solving linear equations. The technique is too complicated to be introduced here. ⋆ The multi-frontal method is more suitable for parallel implementation, although this is an extremely tricky task. Successful implementations include UMFPack, which is the default sparse linear solver in Matlab, and MUMPS, which works on distributed parallel computers.

The Multi-frontal method does not pose strict requirement on the condition number of the matrix, although extremely ill-conditioned matrix may still fail. The memory consumption of multi-frontal method is currently limiting the size of problems handled by multi-frontal solvers. ⋆⋆

The relative merits of multi-frontal methods and iterative methods would depend on the problem on hand. For problems involving advanced physical models, which usually are difficult to converge, the direct multi-frontal methods are more stable. In many cases, the overall (linear + non-linear) solving time is much less with the multi-frontal methods.

13.4 Condition Number of Shockley Equations

Normally the Jacobian matrix of the discretized Shockley equations are well-conditioned. However, in a few situations, ill-conditioning can occur.

Highly non-linear terms in the Shockley equations tend to produce large eigenvalues of the Jacobian matrix. For examples, high concentration of mid-gap charge traps will produce a highly non-linear term. Some advanced mobility models have similar effect as well.

Highly anisotropic mesh is known to increase the disparity between large and small eigenvalues.

Incorrect boundary condition setting almost certainly leads to singular Jacobian matrix. At least one ohmic contact is required to uniquely determine the variables in a device.

Devices with floating regions, such as PNPN junction and partially-depleted SOI devices, have extremely small eigenvalues, and the Jacobian matrices are poorly conditioned.

When there is convergence problem with linear solver, one can try one of the followings to improve the situation

1. Check if the boundary conditions are properly set.
2. Check if there is mesh elements with high aspect ratios, or small angles. Improving mesh quality is often the most effective way to improve convergence.

⋆ Interested readers can check check out the tutorial paper by J.W.H. Liu, "The multifrontal method for sparse matrix solution: theory and practices", SIAM Review 34, pp.82-109 (1992)

⋆⋆ Problems up to 100k unknowns, >1M non-zeros worked on my laptop.

3. Use advanced physical models only when it is important in the study. Disable unnecessary or inappropriate physical models. For example, it is usually not necessary to use surface corrected mobility models to study DIBL in MOSFETs.
4. For devices known to have difficulty in convergence, use direct linear solvers instead of iterative solvers.
5. For devices known to have difficulty in convergence, simulate in transient mode instead of steady-state mode.

A Reading Mathematical Expressions

$a + b$	a plus b
$a - b$	a minus b
$a \times b$	a times b; a multiplied by b
$\frac{a}{b}$	a over b; a divided by b; use b to divide a
$\frac{1}{2}$	one-half
$\frac{1}{3}$	one-third
$\frac{1}{4}$	one-quarter
x^2	the square of x; x-square; squared x
x^3	the cubic of x; x-cube; cubic x
x^4	x to the fourth; x raised to the fourth power
e^x	e-x; e to the power x; e raised to the power x
x_1	x-one; x subscript one
x_0	x-naught; x subscript zero
$f(x)$	f-x; f as a function of x
$\mathrm{d}x$	d-x
$\sin(x)$	sin-x
$\log(x)$	log-x
$\frac{\mathrm{d}f}{\mathrm{d}x}$	d-f d-x; derivative of f with respect to x
$\frac{\partial f}{\partial x}$	partial-f partial-x; partial derivative of f with respect to x

$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$	second derivative of f with respect to x
$\lim_{x \to 0} f(x)$	the limit of f-x as x approaches zero
$\int_a^b f(x)\,\mathrm{d}x$	f-x integrated over x from a to b
$\vec{v}$	v vector; vector v
$\hat{x}$	x hat; unit vector x;
$\vec{a} \cdot \vec{b}$	a dot b
$\vec{a} \times \vec{b}$	a cross b
∇u	gradient of u
$\nabla \cdot \vec{V}$	divergence of V
$\nabla \times \vec{V}$	curl of V
$\boldsymbol{M}$	matrix M
$\boldsymbol{M}_{i,j}$	M-i-j; the element i-j of matrix M
$\boldsymbol{M}^T$	v-transposed; the transpose of vector v
$\boldsymbol{M}^{-1}$	M-inversed; the inverse of matrix M

B Some Mathematics Recapitulation

In this chapter, we shall review a few mathematical concepts useful in this course, namely some terminologies related to partial differential equations. ⋆

B.1 Partial differential equations

Let's look at the Poisson's equation as an example of PDEs,

$$\nabla^2\phi = -\frac{\rho}{\varepsilon}, \tag{B.1}$$

which in 2D Cartesian coordinates can be written as

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = -\frac{\rho}{\varepsilon}. \tag{B.2}$$

B.1.1 PDEs, Terminologies and Classifications

B.1.1.1 Linear vs non-linear

linear:

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} = 0 \tag{B.3}$$

If both u_1 and u_2 satisfy the equation, any linear combination of the two

$$v = \alpha u_1 + \beta u_2 \tag{B.4}$$

also satisfies the equation, where α and β are arbitrary complex constants.

non-linear:

$$au\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} = 0 \tag{B.5}$$

B.1.1.2 homogeneous vs inhomogeneous

Zero RHS vs non-zero RHS.

B.1.1.3 Order

The most general second-order linear PDEs (in 2D), can be written as

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x\partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu = R(x,y) \tag{B.6}$$

⋆ This is not intended to be a comprehensive course, nor is it a tutorial. I only hope to recall those terms and concepts from your memory.

One can easily see its resemblance to the 2nd-order equations that represent conic curves. Indeed, second-order PDEs are commonly classified as elliptic, parabolic and hyperbolic equations. Some examples of homogeneous 2nd order equations are particularly important. ★

- elliptic equations 2D Laplace equations:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{B.7}$$

- parabolic equations: 1D diffusion equations:

$$k\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \tag{B.8}$$

- hyperbolic equations: 1D Wave equations:

$$\frac{\partial^2 u}{\partial x^2} - \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = 0 \tag{B.9}$$

B.1.1.4 Boundary conditions

- Dirichlet: the value of u is specified at each point on the boundary.
- Neumann: the value of $\partial u/\partial n$ (normal derivative) is specified at each point on the boundary.
- Cauchy: both u and and $\partial u/\partial n$ are specified.

B.1.2 Common analytic techniques

- Separation of variables
- Integral transform method
- Green's function method
- Conformal mapping
- Variational method

B.1.2.1 A worked example

Consider the Lapalace equation in 2D,

$$\nabla^2 u = 0. \tag{B.10}$$

The *general solution* of this equation is

$$u = f(x + iy) + g(x - iy), \tag{B.11}$$

for arbitary functions f and g. ★★

★ The determinant is $B^2 - 4AC$.

★★ Verify this.

A convenient choice would be the exponential function, and with some arithmetics, we see that ★

$$v_n = \exp\left(\frac{n\pi y}{a}\right)\sin\left(\frac{n\pi x}{a}\right) \tag{B.12}$$

$$w_n = \exp\left(-\frac{n\pi y}{a}\right)\sin\left(\frac{n\pi x}{a}\right),\ n = 1, 2, 3, \ldots \tag{B.13}$$

both satisfy **(B.10)**. ★★ Further, any linear combinations of v_n and w_n will satisfy **(B.10)** as well. However, this general solution is not good for practical use.

We need to specify the boundary conditions of this problem to proceed further from this general solution, and then find a *particular solution*. In the problem domain $0 \le x \le a, 0 \le y \le b$, let

$$u|_{x=0} = 0, \qquad u|_{(x=a)} = 0,$$
$$u|_{y=0} = x(a-x), \qquad u|_{(y=b)} = x(a-x).$$

This Dirichlet boundary condition is plotted in **Figure B.I**.

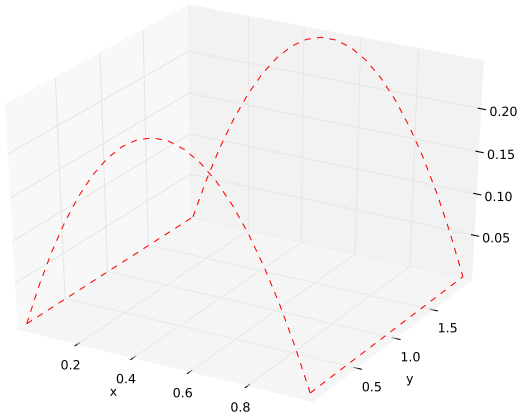


Figure B.I step 0, boundary condition.

As a first attempt, we substitute $n = 1$ in **(B.13)**, scale it and compare with the boundary condition at $y = 0$, which is shown in **Figure B.II**. The match seems to be pretty close, though not exact. Similarly we can add in **(B.12)** to match the BC at $y = b$, which is shown in **Figure B.III**. For many practical purposes, this approximate solution is good enough.

However, if we want an exact solution, we proceed further. Express the general solution as

$$u(x, y) = \sum_{n=0}^{\infty} A_n v_n + B_n w_n.$$
$$= \sum_{n=0}^{\infty} \left[A_n \exp\left(\frac{n\pi y}{a}\right) + B_n \exp\left(-\frac{n\pi y}{a}\right)\right]\sin\left(\frac{n\pi x}{a}\right) \tag{B.14}$$

At the boundaries $y = 0$ and $y = b$, the solution must match the BC,

---

★ recall that $\sin x = \left(e^{jx} + e^{-jx}\right)/2$, $\cos x = \left(e^{jx} - e^{-jx}\right)/2j$

★★ I also conveniently cheated here, but in what way?

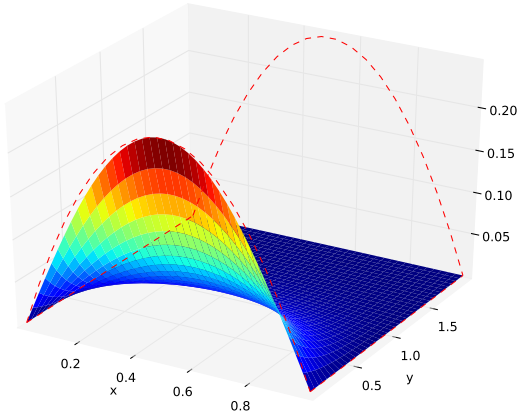


Figure B.II step 1, comparison of $0.25w_1(x,y)$ with the BC.

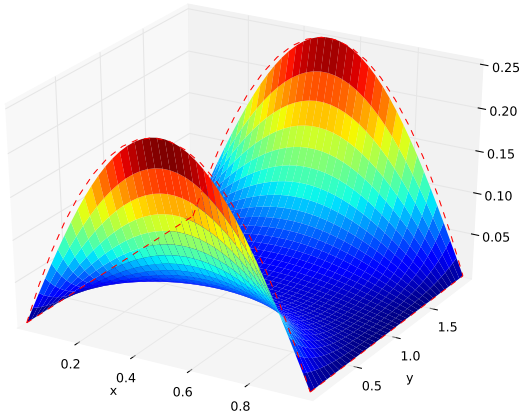


Figure B.III step 2, an approximate solution.

$$\sum_{n=0}^{\infty}\left[A_n+B_n\right]\sin\left(\frac{n\pi x}{a}\right)=x(a-x), \tag{B.15}$$

$$\sum_{n=0}^{\infty}\left[A_n\exp\left(\frac{n\pi b}{a}\right)+B_n\exp\left(-\frac{n\pi b}{a}\right)\right]\sin\left(\frac{n\pi x}{a}\right)=x(a-x). \tag{B.16}$$

We can find the coefficients A_n and B_n with Fourier series expansion. ★

$$u(x,y)=\frac{8a^2}{\pi^3}\sum_{n=1}^{\infty}\frac{\cosh\left[(2k-1)\pi(y-b/2)/a\right]}{(2k-1)^3\cosh\left[(2k-1)\pi b/2a\right]}\sin\frac{(2k-1)\pi x}{a} \tag{B.17}$$

It is obvious that the coefficients of high-order terms in **(B.17)** diminishes quickly ($\sim k^{-3}$).

★ A tedious procedure.

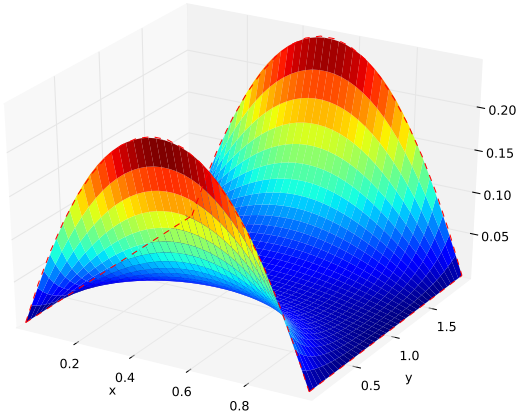


Figure B.IV step 3, first 3 term in the series solution **(B.17)**.